E12

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 :
B01J 8/02, C07K 1/04, 1/06, 1/08, 1/10, C08F 283/00, C08G 69/10, G05B 13/00, 19/402

A1

(11) International Publication Number: WO 96/22157

(43) International Publication Date: 25 July 1996 (25.07.96)

(21) International Application Number: PCT/US96/01168

(22) International Filing Date: 19 January 1996 (19.01.96)

(30) Priority Data:
375,879 20 January 1995 (20.01.95) US

(71) Applicant: SELECTIDE CORPORATION [US/US]; 1580 East Hanley Boulevard, Tucson, AZ 85737 (US).

(72) Inventors: KRCHNAK, Viktor; 10700 N. La Reserve Drive #16206, Tucson, AZ 85737 (US). LEBL, Michal; 12460 North Granville Canyon Way, Tucson, AZ 85737 (US). SELIGMANN, Bruce; 6290 N. Nirvana Place, Tucson, AZ 85715 (US).

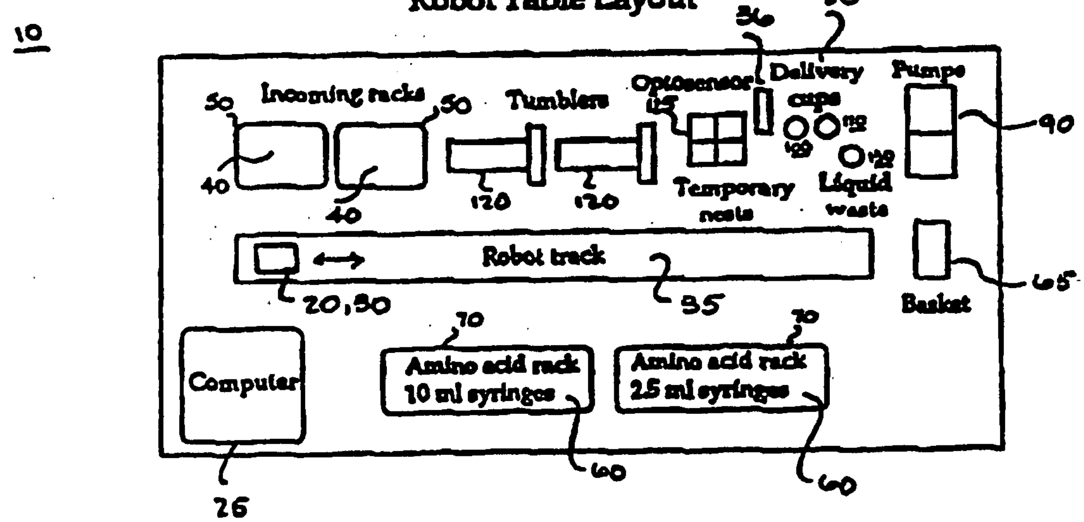
(74) Agents: BALDWIN, Geraldine, F. et al.; Pennie & Edmonds, 1155 Avenue of the Americas, New York, NY 10036 (US).

(81) Designated States: AL, AM, AU, AZ, BB, BG, BR, BY, CA, CN, CZ, EE, FI, GE, HU, IS, JP, KG, KP, KR, KZ, LK, LR, LS, LT, LV, MD, MG, MK, MN, MX, NO, NZ, PL, RO, RU, SG, SI, SK, TJ, TM, TR, TT, UA, UZ, VN, ARIPO patent (KE, LS, MW, SD, SZ, UG), Eurasian patent (AZ, BY, KG, KZ, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: APPARATUS AND METHOD FOR MULTIPLE SYNTHESIS OF ORGANIC COMPOUNDS ON POLYMER SUPPORT

Robot Table Layout



(57) Abstract

A solid phase synthesis system is provided by employing a fully automated robot (20) that operates with a novel timing protocol executed by a computer (25) for handling multiple synthetic tasks efficiently. The fully automated robot (20) moves along a track (35) and is equipped with a gripper arm (30) which can pick up, position, and operate syringes (40, 60) which can contain solid supports and amino acid reactants. The novel timing protocol is realized by performing steps in the synthesis cycles for different compounds, such as peptides, concurrently rather than on a sequential basis.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AM	Armenia	GB	United Kingdom	MW	Malawi
AT	Austria	GE	Georgia	MX	Mexico
AU	Australia	GN	Guinea	NE	Niger
BB	Barbados	GR	Greece	NL	Netherlands
BE	Belgium	HU	Hungary	NO	Norway
BF	Burkina Faso	IE	Ireland	NZ	New Zealand
BG	Bulgaria	IT	Italy	PL	Poland
BJ	Benin	JP	Japan	PT	Portugal
BR	Brazil	KE	Kenya	RO	Romania
BY	Belarus	KG	Kyrgystan	RU	Russian Federation
CA	Canada	KP	Democratic People's Republic of Korea	SD	Sudan
CF	Central African Republic	KR	Republic of Korea	SE	Sweden
CG	Congo	KZ	Kazakhstan	SG	Singapore
CH	Switzerland	LI	Liechtenstein	SI	Slovenia
CI	Côte d'Ivoire	LK	Sri Lanka	SK	Slovakia
CM	Cameroon	LR	Liberia	SN	Senegal
CN	China	LT	Lithuania	SZ	Swaziland
CS	Czechoslovakia	LU	Luxembourg	TD	Chad
CZ	Czech Republic	LV	Latvia	TG	Togo
DE	Germany	MC	Monaco	TJ	Tajikistan
DK	Denmark	MD	Republic of Moldova	TT	Trinidad and Tobago
EE	Estonia	MG	Madagascar	UA	Ukraine
ES	Spain	ML	Mali	UG	Uganda
FI	Finland	MN	Mongolia	US	United States of America
FR	France	MR	Mauritania	UZ	Uzbekistan
GA	Gabon			VN	Viet Nam

APPARATUS AND METHOD FOR MULTIPLE SYNTHESIS OF ORGANIC COMPOUNDS ON POLYMER SUPPORT

Technical Field

This invention relates to an apparatus and method for the multiple synthesis of compounds consisting essentially of repeating structural units formed, for example, by repeated washing, deprotection, and coupling. More particularly, the invention is directed to a system for the multiple synthesis of organic compounds, such as peptides, which uses an automated robot for handling multiple synthetic tasks.

Reference To Microfiche Appendix

A microfiche appendix consisting of two (2) microfiches and of a hundred twenty (120) frames is included as a part of the specification. The microfiche appendix contains a program listing implementing the present invention. The program listing is subject to copyright protection. The copyright owner, however, has no objection to the facsimile reproduction by anyone of the program listing, as it appears in the Patent and Trademark Office file or records, but otherwise reserves all copyright rights whatsoever.

Background of The Invention

Systems for the synthesis of organic compounds, for example peptides, are highly desired for many applications. For example, the synthesis and collection of a large number of peptides would assist in the development of agents that could block, promote or otherwise affect cellular reactions that involve recognition and binding. These agents would be useful in the treatment or diagnosis of a number of diseases.

More particularly, synthetic peptides can be used as diagnostic and therapeutic agents.

Understandably, peptide synthesis systems have been designed and constructed. Houghten. R.A., Proc. Natl Acad. Sci., 82: 5131-5135 (1985), employs a "tea bag" method using standard Boc amino acid resin in polypropylene mesh packets with standard washing, deprotection, neutralization, and coupling protocols of the original solid phase procedure of Merrifield, R.B., J. Amer. Chem Soc., 85: 2149-2154 (1963).

Although some peptide synthesis systems have been automated for the synthesis of multiple peptides, they generally exhibit poor "respite" time, unable to handle efficiently multiple synthetic tasks. For example, commercial peptide synthesis systems, such as those from Gilson, USA and Advanced Chem Tech, are capable of synthesizing multiple peptide sequences. However, these automated synthesis systems perform each step in the synthetic cycles for all peptides one by one or sequentially. The timing protocol, more specifically, involves washing all peptides one by one, deprotecting all peptides one by one and then coupling all peptides one by one. For example, coupling is initiated for a first peptide and then when those first initiation coupling steps are completed, coupling is initiated for a second peptide. Such a timing protocol results in an especially long delay or respite time between peptide synthesis steps. Washing all peptides one by one is time consuming. Moreover, during coupling, the system is idle while waiting for coupling to finish. With coupling taking up to 2 hours, the efficiency of prior art automated synthesis systems is severely limited.

Furthermore, prior art automated peptide synthesis systems critically lack flexibility. Once the synthesis of a peptide set has started, any additional peptides, even those urgently required, cannot be started. Furthermore, prior art systems lack the flexibility so as to execute more than one type of coupling reaction, which is typically required for the synthesis of non-peptide compounds.

Summary of The Invention

A novel synthesis system overcoming the drawbacks of the prior art is realized by employing a fully automated robot that operates with a novel timing protocol for handling multiple synthetic tasks efficiently. The novel timing protocol is realized by performing different steps in the synthesis cycles for multiple organic compounds concurrently rather than on a sequential basis.

Advantageously, this novel timing protocol provides the flexibility of readily adding desired compounds to the list of compounds to be synthesized, changing the order in which the compounds are synthesized, or deleting previously entered compounds. More importantly, such a timing approach or protocol decreases the "respite" time - new synthesis steps of the next compound synthesis being initiated prior to the completion of those in the previous synthesis.

In an exemplary embodiment, the novel synthesis system includes an industrial robot system operating under computer control for effecting the synthesis steps within each cycle of the selected compounds, for example, peptides. Each synthetic cycle comprises five different steps: washing, adding deprotection

reagents, deprotection, adding coupling reagents and coupling. Because deprotection and coupling do not involve any robot action they are passive synthetic steps, although chemically active; the other steps are active synthetic steps.

Importantly, the novel timing protocol is realized by having the robot system, during the passive synthetic steps of a compound synthesis, perform active synthetic steps for the synthesis cycle of the synthesis of the next compound.

In a preferred embodiment of the invention, a solid-phase peptide synthesis is used, with a first set of plastic syringes containing the resin used as the solid support and serving as the reaction vessel. Additionally, a second set of plastic syringes contains amino acids. The robot system equipped with a gripper arm positions any one of the plurality of syringes to selectively aspirate and dispense solvents and reagents from the desired syringe in accordance with the novel timing protocol for coupling predetermined amino acids to the solid support.

Brief Description of The Drawings

A more complete understanding of the invention may be obtained by reading the following description in conjunction with the appended drawing in which like elements are labeled similarly and in which:

Fig. 1 is a block diagram of a synthesis system in accordance with the principles of the invention;

Figs. 2A and 2B are illustrative flow charts representing the operation of the synthesis system of Fig. 1, and more particularly, the timing protocol; and

Fig. 3 is a chart illustrative of the timing protocol for the synthesis system of Fig. 1.

Detailed Description

The invention relates to an apparatus for the multiple synthesis of compounds consisting essentially of repeating structural units, for example peptides, formed by repeated washing, deprotection, and coupling. The apparatus or synthesis system employs a fully automated robot system that operates with a novel timing protocol for handling multiple synthetic tasks efficiently. The novel timing protocol is realized by performing different steps in the different synthesis cycles of multiple organic compounds concurrently rather than on a sequential basis, as discussed in more detail below.

Without any loss of generality or applicability for the principles of the present invention, the overall operation of the present invention is described using, as an example, Fmoc/tBu chemistry, applied to the Merrifield solid phase synthesis of peptides. It should, however, be understood that the present invention is equally applicable to any other chemistry for the solid phase synthesis of compounds that consist essentially of repeating structural units, such as polymers or organic compounds as described, for example, in Borchardt et al., J. Am. Chem. Soc., 116: 373 (1994); Chen et al., J. Am. Chem. Soc., 116: 2661 (1994); Simon et al., PNAS, 89: 9367 (1992); Nikolaiev et al., Pept. Res., 6: 161-70 (1993); and Lebl et al., Vol. 5: 541-548, Techniques In Protein Chemistry, Academic Press, San Diego (1994), which are incorporated herein by reference.

Before describing the present invention, however, it will be instructive to discuss briefly the basic principles of peptide synthesis. Peptide synthesis involves the coupling of amino acids and may be accomplished by techniques familiar to those skilled in the art. See, for example, Stewart and Young, Solid Phase Synthesis, Second Edition, Pierce Chemical Co., Rockford, IL (1984), which is incorporated herein by reference.

The process of peptide synthesis on solid supports generally involves building a peptide from the carboxyl or C-terminal end in which the C-terminal amino acid with its protected α-amino group is attached to a solid phase polymer. The protecting group is then cleaved off, and the next amino acid, also protected, is coupled by a peptide bond to the α-amino group of the amino acid attached to the solid support. The cycle of deprotecting the prior amino acid and coupling the additional amino acid is repeated until the desired peptide is synthesized. Any reactive side chains of the amino acids are protected by chemical groups that can withstand the coupling and Nα-deprotection procedure. These side chain protecting groups, however, can be removed at the end of the synthesis.

In order to couple an amino acid to the growing synthetic chain, the carboxyl group of the blocked amino acid must be activated. Many methods of activation may be used, including, for example, preformed symmetrical anhydrides (PSA), preformed mixed anhydride (PMA), acid chlorides, active esters, and *in situ* activation of the carboxylic acid.

The present synthesis system includes a fully automated robot system interfaced to a computer for effecting the synthesis cycle of each desired peptide to be synthesized. Referring to Fig. 1, there is illustrated a block diagram of a preferred embodiment of a system for the multiple synthesis of compounds, more particularly peptides, applying standard Fmoc/tBu chemistry in accordance with the principles of the invention.

Synthesis system 10 comprises an industrial robot system 20, such as Model A251 from CRC Plus, Inc., Harrington, Canada, that is interfaced to a IBM personal computer 25. Computer 25 includes such hardware as a central processing unit, program and random access memories, timing and control circuitry, input and output interface devices, and other digital subsystems necessary to the operation of the central processing unit, all of which are well known and understood in the art. Accordingly, such computer hardware is not discussed herein for the sake of clarity.

Robot system 20 is programmed to effect different mechanical manipulations through the use of pre-programmed steps typically provided for in software programs delivered with the robot system. Alternatively, such preprogrammed steps can be provided for in third party software programs, such as Total Control For Windows, Hudson Control Group Inc., Springfield, New Jersey. Those skilled in the art will readily note that other computer means, microcomputer control means or other automated control means, including other programming software, may be

alternatively provided to effect mechanical manipulations.

Automated robot system 20 equipped with a gripper arm 30 that is movable under program control along track 35 can pick up and position (X,Y,Z) any one of a plurality of syringes to any predetermined location. The gripper arm is used to selectively aspirate and dispense solvents and reagents from the desired syringe which serves as the reaction vessel for coupling predetermined amino acids to a solid support. More particularly, each syringe has a plunger that is movable by the gripper arm so as to aspirate or dispense reagents in a manner well known to those skilled in the art. See, for example, Krchnak, V., Vagner, J., "Color-Monitored Solid-Phase Multiple Peptide Synthesis Under Low-Pressure Continuous Flow Conditions," Peptide Res. 3, 182 (1990), which is incorporated herein by reference. Moreover, the gripper arm provides a means for shaking the contents of the syringes. Also, the content amount of reagents in each syringe can be readily determined by detecting the position of the plunger within the desired syringe. Also, to confirm that the syringe has been properly gripped and positioned, an optical sensor 36 detecting the motion of the syringes is preferably used to provide feedback to the synthesis system.

A suitable solid support may be selected, such as a polystyrene or polyethylene glycol/polystyrene resin. The protected amino acids may be readily obtained from a number of places, such as Bachem (Torrance, California), Advanced ChemTech (Louisville, Kentucky) or Propeptide (Vert-le-Petit, France).

Importantly, computer 25 also controls when each step in the synthesis cycle is initiated so as to implement the novel timing protocol of the present invention wherein different steps in the synthesis cycles of multiple peptides are performed concurrently rather than on a sequential basis.

More particularly, synthesis system 10 includes a first set of plastic syringes 40 for holding resin used as the solid support, herein referred to as "resin syringes." In the preferred embodiment, RAM-TentaGel (0.21 mmol/g) obtained from Rapp-Polymere, Tubingen, Germany, is used as the resin.

First set of plastic resin syringes 40 is placed in incoming racks 50, each holding up to, for example, thirty 10 ml and thirty 2.5 ml syringes. Synthesis system 10 also includes a second set of plastic syringes 60 for holding amino acid solutions, herein referred to as "amino acid syringes." Similarly, second set of syringes 60 are placed in racks 70, each holding up to, for example, a hundred 10.0 ml and a hundred 2.5 ml syringes. Basket 65 holds the completed synthesized peptides. The number and size of the first and second sets of syringes will be dependent on the number of different peptides synthesized, the length of the peptides and the number of different amino acids required.

Plastic syringes 40, 60 preferably are made of a material, such as polypropylene, that is sufficiently chemically inert to all solvents and reagents used in the solid-phase peptide synthesis, including trifluoroacetic acid. Alternative materials include Kevlar, Teflon or cast glass.

Each plastic resin syringe 40, which serves as the reaction vessel, is equipped at the bottom with a frit (not shown). It should be understood that various other vessels and means for retaining the solid phase and removing the excess reagents and solvents may be used. For example, columns or wells fitted with a frit or filter may be used as alternative reaction vessels that are compatible with the present synthesis system. In that alternative, the columns or wells are connected to a vacuum source in such a manner that they can be disconnected from the reaction vessel. Moreover, the frit or the means for retaining the solid phase does not have to be integral with the reaction vessel. Rather, it can be external or only present when necessary to retain the solid phase during the removal of solution therefrom.

Appropriate solvents and reagents are introduced from solvent reservoirs to delivery cups 80 using four 10 ml piston pump systems 90, such as the Hamilton MicroLab 900, Reno Nevada. Commercial-grade solvents and reagents, such as dimethylformaminde (DMF), piperidine/DMF, DIC/DMF, are used. Three of the piston pumps deliver solvents and reagents to delivery cups 80 whereas the fourth piston pump removes waste as well as wash solution from cups 80 where the amino acids and the coupling reagents are mixed.

Delivery cups 80 comprise a first cup 100 that is used for all washing and for receiving the protected amino acids and coupling reagents, and a second cup 110 that is used for receiving deprotection reagents.

Those skilled in the art will readily note that further independent manipulating stations may be used

to manipulate the syringes, depending on the particular reaction desired therein, such as dispensing, aspirating, shaking, heating, cooling or even refluxing. In those instants, robot system 20 would only be required to position the appropriate syringes in the desired manipulating stations. This would decrease the required time to perform the active synthetic steps because the most time consuming tasks would be distributed among multiple and independent manipulating stations.

Using the above synthesis system, multiple peptides are synthesized by manipulating the syringes so as to repeatably attach during a synthetic cycle a desired amino acid to the solid support. Those skilled in the art will note that there is neither a physical link, such as tubing and valves, between the reagent vessels and the reaction vessels nor between the mixing chambers and the reaction vessels. As such, different reagents and reaction conditions may be used within the same or different peptide synthesis.

Each synthetic cycle consists of the following sequence of basic operations or steps: washing, adding deprotection reagents, deprotection, adding coupling reagents, and coupling. Judiciously combining synthetic cycles, each coupling a desired amino acid, creates a synthesis protocol for the synthesis of the desired peptide. Only three of the above synthetic steps, washing, adding deprotection reagents and adding coupling reagents, however, directly require the use of robot system 20. These steps are referred herein to as "active synthetic steps," although

chemically passive, and are discussed in more detail below.

During washing, robot system 20 positions a selected resin syringe from rack 40, a tumbler 120 or holding position 125 and then dispenses any liquid therein into liquid waste 130. A preprogrammed amount of washing liquid is subsequently delivered to first cup 100 using piston pump system 90 and then aspirated together with a preprogrammed amount of air using the selected resin syringe. Robot system 20 finally then shakes the selected resin syringe for a preprogrammed time to ensure efficient mixing.

Adding deprotection reagents similarly requires the use of robot system 20. Robot system 20 positions the desired resin syringe over liquid waste 130 and dispenses any liquid therein. A preprogrammed amount of deprotection mixture is subsequently delivered to second cup 110 and then aspirated together with a preprogrammed amount of air using the selected resin syringe.

Lastly, adding coupling reagents also requires the use of robot system 20. Robot system 20 positions the selected resin syringe in holding position 125. Robot system 20 then selects and positions a selected amino acid syringe, dispensing a predetermined amount of desired amino acid into first cup 100. The selected amino acid syringe is repositioned in rack 40. A preprogrammed amount of activating agents is then delivered to first cup 100 and aspirated together with a preprogrammed amount of air using the selected resin syringe located in holding position 125. That selected resin syringe is then positioned into tumbler

120. Next, robot system 20 dispenses a preprogrammed amount of washing solvent into first cup 100 which is then aspirated. Typically, the latter steps of tumbling and washing are repeated a number of times, preferably three times.

During deprotection and coupling, the syringes are placed on 2.5 ml or 10.0 ml tumbling racks 120, each of which holds up to 18 syringes, to ensure an efficient mixing.

In order to better understand the timing protocol that allows the handling of multiple synthetic tasks more efficiently, a discussion of the underlying basis for the timing protocol would be instructive, particularly with reference to the flow charts of Figs. 2A and 2B.

To begin with, the user enters the sequence of compounds to be synthesized. And, if the necessary reagents are available, the system will start to initiate a desired synthesis - otherwise, the user is notified of the deficiencies. More particularly, for each desired synthesis, the system determines the particular synthetic steps, both active and passive, that are required to synthesize the sequence as well as determines when those synthetic steps are to be performed - that is, their timing or temporal relationship within the synthetic cycle.

Those skilled in the art will readily note that these synthetic steps are sequence specific, that is they are specifically correlated to the desired synthesis. Herein, these synthesis steps are referred to as a "sequence specific timing protocol." In

considering to initiate a new synthesis, the "sequence specific timing protocol" of the new desired synthesis is compared to the sequence specific steps or timing protocol for those sequences already initiated. These latter sequence specific steps are referred to as a "cumulative remaining timing protocol" because they are a compilation of all the individual "sequence specific timing protocols" of currently initiated syntheses.

More particularly, the above comparison requires the system to plot out when the active synthetic steps of the new desired synthesis occur with respect to the active synthetic steps of the sequences currently being synthesized. And, unless there is a timing conflict in performing the active synthetic steps of the new desired synthesis and those already initiated, the system begins to initiate the start of the new synthesis, if necessary reagents are available. It should be understood that the new sequence specific timing protocol of the desired new synthesis is added to the "cumulative remaining timing protocol."

Should, however, a timing conflict exist, the system attempts to determine if the initiation of the new desired synthesis can be delayed or "time shifted", up and until the start of the next active synthetic step of any of the initiated syntheses, so as to eliminate the timing conflict. Unless the system is able to resolve the timing conflict, the system will not initiate the new desired synthesis, but will instead proceed to execute the next required active synthetic step of the syntheses currently in progress. Once the active synthetic step has been performed, the system updates the "cumulative

remaining timing protocol" and then attempts again to resolve the timing conflict during the next passive synthetic step of any of the ongoing syntheses. This timing protocol is repeatably performed until all the desired syntheses have been initiated.

Below is an illustrative description of the synthetic steps for the simultaneous synthesis of multiple peptides in accordance with the principles of the timing protocol discussed above. It should be recalled that each synthetic protocol in the multiple synthesis is a sequenced cycle order of pre-programmed steps or procedures. For this illustrative description, one synthetic cycle consists of the following synthetic steps: (i) washing the resin three times with dimethylformamide (DMF) for thirty (30) seconds each time; (ii) deprotecting the amino acid with fifty percent (50%) solution of piperidine/DMF for ten (10) minutes; (iii) washing the resin five (5) times with dimethylformamide (DMF) for thirty (30) seconds each time; and (iv) two (2) hour coupling using three (3) molar excess of DIC/HOBt activated Fmoc protected amino acids.

After finishing the last synthetic cycle for each peptide, the synthesis is finished by washing, or by washing, deprotecting and washing, if that is applicable.

Robot system 30 operating in accordance with the above illustrative synthetic cycle positions a first resin syringe (peptide synthesis no. 1), manipulates the plunger to aspirate a wash solution of dimethylformamide (DMF), and then shakes the resin syringe. This washes the resin with wash solution for

thirty (30) seconds. Preferably, the washing is repeated three times. Subsequently, the first resin syringe is used to aspirate from the delivery cup the deprotection reagents that have been dispensed by the piston pump. This latter step is the "adding deprotection reagents" step. This repeated washing and adding deprotection reagents takes about five (5) minutes.

During deprotection, which lasts about ten (10) minutes, the synthesis system can advantageously direct the robot system to initiate washing and adding deprotection reagents for a second resin syringe or the next peptide synthesis (peptide synthesis no. 2). Since that washing and adding deprotection reagents, however, only takes less than five minutes, there are at least five minutes still remaining before the deprotection is completed for the first resin syringe. To ensure that the respite time between steps is minimized, the synthesis system further directs the robot system to initiate yet another washing and adding deprotection reagents for a third resin syringe (peptide synthesis no. 3).

Determining whether to initiate another washing and adding deprotection reagents for a subsequent resin syringe (the next peptide synthesis) is controlled automatically under program control by comparing the sequence specific timing protocol with the cumulative remaining timing protocol, as discussed above. In effect, washing and adding deprotection agents are initiated if the least remaining time for deprotection is sufficiently long to complete the task.

Once deprotection has been completed for the first resin syringe, the synthesis system directs the robot system to position the first resin syringe so as to wash the resin for thirty seconds with dimethylformamide (DMF). This is repeated, preferably, five times, after which the resin syringe is positioned in a temporary holding location. The amino acid syringe containing the desired amino acid to be coupled is next positioned and a preprogrammed amount dispensed into the delivery cup which then receives a DIC solution. The first resin syringe is used to aspirate from the delivery cup the amino acid solution so as to couple that amino acid. This latter step is the "adding coupling reagents" step. Likewise, the time for washing and adding coupling reagents takes about five (5) minutes. During that coupling, the synthesis system advantageously directs the robot system to initiate coupling for the second resin syringe (peptide synthesis no. 2), and then subsequently for the third resin syringe (peptide synthesis no. 3).

Similarly, determining whether to initiate another washing and add coupling reagents for a subsequent resin syringe or peptide synthesis is controlled automatically under program control by comparing the sequence specific timing protocol with the cumulative remaining timing protocol, as discussed above. In effect, washing and adding coupling reagents are initiated if the least remaining time for coupling is sufficiently long to complete the task.

Once coupling has been initialized for all three resin syringes, that is the three peptide syntheses, the synthesis system takes a fourth resin syringe

(peptide synthesis no. 4) and initiates washing and deprotection. The synthesis system initiates washing and deprotection for as many resin syringes (peptide syntheses) as possible before the least remaining coupling time expires.

This procedure is repeated until the sequence of the shortest length peptide is assembled. If desired, an additional resin syringe can then be used to synthesize a new peptide sequence.

The maximum amount of peptide synthesized is limited by the size of the syringe. For example, with 10 ml syringes containing about 500 mg of resin (0.5 mmol/g resin), it is anticipated that 0.25 mmol of peptide (ca 300 mg crude decapeptide) can be synthesized.

Those skilled in the art will readily note in the above illustrative description that the coupling time for each peptide synthesis is the same - the earliest initiated coupling finishing prior to subsequently initiated couplings. In that situation, each new initiated synthetic cycle for a peptide synthesis follows the order in which the prior cycle was initiated. Of course, it should be understood that different coupling times, if that is desired, may be used. The principles of the timing protocol, however, do not change - the system, if time permits, initiates an active synthesis step of the next peptide synthesis during the passive steps of the prior peptide synthesis.

A program listing implementing the principles of the invention, including the timing protocol, is contained in the microfiche appendix.

Referring to Fig. 3, there is illustrated the timing protocol or timing relationship between washing, adding deprotection reagents, deprotection, adding coupling reagents and coupling for the multiple synthesis of twelve peptides using the above described solid phase peptide synthetic protocol. Those skilled in the art will readily note based on the discussion above that during passive synthetic steps (deprotection and coupling), active synthetic steps in the synthetic cycles of the next peptide synthesis are concurrently initiated with the prior peptide synthesis. For example, referring to Fig. 3, during deprotection of the first peptide synthesis (peptide synthesis no. 1), washing and adding deprotection reagents is initiated for the second and third peptide syntheses (peptide synthesis nos. 2 and 3).

It should be understood that prior to initiating the synthesis of each compound, various parameter variables must be specified, in the synthetic protocol of each desired peptide synthesis, including the kind and amount of solvent or reagents to be dispensed, and the length of time of the corresponding washing steps and/or reaction steps. Also, because aspiration is performed in adding deprotection reagents and adding coupling reagents, the volume of air necessary to enable the efficient mixing of the resin and solvent must be specified.

In addition to the various parameter variables which must be specified or retrieved from existing

stored synthetic protocols, the synthesis system must also be initialized with other information, including the assignment and identification of solvents and reagents contained in the peptide synthetic protocols; the reagent and amino acid concentrations (mmol/ml) used to calculate the corresponding reagent and amino acid solution volumes, respectively; the size of the syringes; the amount of resin in the syringes; the substitution of resin in mmol/g; the positions and content of each amino acid syringe, preferably identified by a three to five letter code, including its concentration and available volume; the peptide sequences desired to be synthesized using the identifying three to five letter codes; and for each peptide its identifying code, and sequence. Preferably, the system prompts the user for the information. Alternatively, however, this information may be entered prior to initiating the peptide syntheses or stored in a file or database.

The order in which sequences to be synthesized are entered establishes their priority for synthesis, and establishes a synthesis queue. Advantageously, any compound can be entered with any priority. Also, additional sequences can be added to the synthesis queue together with the necessary synthetic protocol. Also, additional reagents may be added, if necessary.

From the above information, the system calculates whether the amount of amino acid and reagents is sufficient to prepare all the peptides. Unless there is enough amino acid and reagents available for the synthesis, the synthesis system will not initiate that synthesis, indicating instead its status as "not ready." Significantly, the above timing protocol

allows handling multiple synthetic tasks efficiently since active synthetic steps requiring the use of the robot system in the synthesis cycles of multiple peptides are performed concurrently with passive synthetic steps in different synthetic cycles. Moreover, because synthesis resources become immediately available with each completed synthesis, a new peptide synthesis can be initiated without waiting for the complete synthesis of the entire set of desired peptides. Preferably, the new peptide synthesis is taken from the top of a waiting list of peptides that has been prioritized according to need. Advantageously, that prioritized waiting list can be changed at any time, either by adding or deleting any peptide, to meet varying situations. Further, any peptide currently being synthesized can be halted without affecting the synthesis of the others.

Advantageously, since there is neither a physical link between the reagent vessels and the reaction vessels nor between the mixing chambers and the reaction vessels, different reagents and reaction conditions may be used within the same or different peptide synthesis. Such flexibility, of course, allows for the synthesis of libraries requiring variable numbers of building block reagents.

To test the reliability and throughput of the above novel peptide synthesis system, five different peptide amides (IKRKR, VRYGI, AAAGY, FPRGR, VYFAW) were synthesized using a RAM TentaGel resin in accordance with the principles of the invention. Also, a mixture K (82.5% trifluoroacetic acid, 5% p-cresol, 5% thioanisole, 2.5% ethanedithiol and 5% water) was used to deprotect side chains and to cleave

the peptides from the resin. Results from that experimental practice indicate, a high degree of purity of the crude material based on analytical gradient HPLC traces and molecular peak analysis using mass spectroscopy.

It is understood that various other modifications will also be readily apparent to those skilled in the art without departing from the scope and spirit of the invention. For example, the principles of the invention can be equally applied to any chemistry that involves repeated procedures, such as washing, deprotection and coupling - particularly, solid phase synthesis, including but not limited to the organic reactions described in Borchardt et al., J. Am. Chem. Soc., 116: 373 (1994); Chen et al., J. Am. Chem. Soc., 116: 2661 (1994); Simon et al., PNAS, 89: 9367 (1992); Nikolaiev et al., Pept. Res., 6: 161-70 (1993); and Lebl et al., Vol. 5: 541-548, Techniques In Protein Chemistry, Academic Press, San Diego (1994).

Accordingly, it is not intended that the scope of the claims appended hereto be limited to the description set forth herein, but rather that the claims be construed as encompassing all the features of the patentable novelty that reside in the present invention, including all features that would be treated as equivalents thereof by those skilled in the art to which this invention pertains.

FOCUS PAGE

Apparatus and Method for Multiple Synthesis of Organic Compounds on Polymer Support

Viktor Krchnak et al.

Apparatus and Method for Multiple Synthesis of Organic Compounds on Polymer Support

Viktor Krchnak et al.

```
;
;  -   Procedure : AMINO
;      Purpose   : Print the current values of amino acid volumes
;      Author    : CPS
;-

disablekeys
filenew 2,"temp.prt"
if (num_codes)
  filewriteline 2,"Required Amino Acid Amounts"
  filewriteline 2,""
  filewriteline 2,"Code  Req. Avail  Code  Req.  Avail  Code  Req.  Avail  Code
  filewriteline 2,""
  a = 1
  while (a <= num_codes && a <= 50)
    a$ = ""
    stringformat b$,"%4.3s %5.1lf %5.1lf  ",aalist_code$[a],aalist_req(a),aalist
    a$ = a$ + b$
    if (a+50 <= num_codes)
      stringformat b$,"%4.3s %5.1lf %5.1lf  ",aalist_code$[a+50],aalist_req(a+50
      a$ = a$ + b$
    endif
    if (a+100 <= num_codes)
      stringformat b$,"%4.3s %5.1lf %5.1lf  ",aalist_code$[a+100],aalist_req(a+1
      a$ = a$ + b$
    endif
    if (a+150 <= num_codes)
      stringformat b$,"%4.3s %5.1lf %5.1lf",aalist_code$[a+150],aalist_req(a+150
      a$ = a$ + b$
    ndif
    .ilewriteline 2,a$
    a = a + 1
  endwhile
  filewriteline 2,ascii$(12)+"Available Amino Acid Amounts"
else
  filewriteline 2,"Available Amino Acid Amounts"
endif
filewriteline 2,""
filewriteline 2,"          10 ml Syringe Rack                          2.5 ml Syringe
filewriteline 2,""
filewriteline 2,"Pos Code Conc  Vol  Pos Code Conc  Vol   Pos Code Conc  Vol  Po
a = 1
while (a <= 50)
  a$ = ""
  stringformat b$,"%3.0lf %4.3s %4.1lf %4.1lf  ",a,acid_code$[a],acid_conc(a),ac
  a$ = a$ + b$
  stringformat b$,"%3.0lf %4.3s %4.1lf %4.1lf   ",a+50,acid_code$[a+50],acid_con
  a$ = a$ + b$
  stringformat b$,"%3.0lf %4.3s %4.1lf %4.1lf  ",a,acid_code$[a+100],acid_conc(a
  a$ = a$ + b$
  stringformat b$,"%3.0lf %4.3s %4.1lf %4.1lf",a+50,acid_code$[a+150],acid_conc(
  a$ = a$ + b$
  filewriteline 2,a$
  a = a + 1
endwhile
```

```
fileclose 2
fileprint "temp.prt"
enablekeys
```

```
;=========================================================
;       Procedure : AMINO
;       Purpose   : Allow operator to change amino acids
;       Author    : MJB,CES
;=========================================================

local a,b,a$,b$,lastcoderow,startindex,sel_row

disablekeys
; Set the colors of the volumes -- set the ones that have their required
; greater than their available to red.
setnumarray code_color[1],15,-1
a = 1
while (a <= num_codes)
  if (aalist_req[a] > aalist_avail[a])
    code_color[a] = 4
  endif
  a = a + 1
endwhile

; Start off by showing all codes
a = 1
while (a <= 100)
  code_index[a] = a
  code_index[a+100] = a
  a = a + 1
endwhile
code_row = 0
lastcoderow = 0
ten_row = 0
ten_col = 0
num_ten = 100
two_row = 0
two_col = 0
num_two = 100

; Store temporary data so we can go back to it
savevars 1,"tmpamino.dat"
savevars 2,"tmpcodes.dat"

screenon "amino"
enablekeys
user_choice = 0

while (user_choice != -1)
  if (user_choice == 1)
    call aminoprt.tcl
    user_choice = 0
  elseif (user_choice == 2)
    ; Clear all amino acid values
    ask user_choice,"Are you sure you want to clear\nall amino data?"
    if (user_choice)
      setstringarray acid_code$[1],"",200
      setnumarray acid_conc[1],0,200
      setnumarray acid_avail[1],0,200
```

-29-

```
      calc_needed = 1
      updatetable
     endif
     user_choice = 0
  elseif (user_choice == 5)
    ask user_choice,"Cancel changes to amino rack?"
    if (user_choice)
      ; Retrieve temporary data
      loadvars 1,"tmpamino.dat"
      loadvars 2,"tmpcodes.dat"
      screenback
      stop
    endif
    user_choice = 0
  elseif (code_row != lastcoderow)
    ; show a subset of the codes
    disablekeys
    screenhold
    if (code_row == 0)
      ; show all codes
      a = 1
      while (a <= 100)
        code_index[a] = a
        code_index[a+100] = a
        a = a + 1
      endwhile
      num_ten = 100
      num_two = 100
    else
      ; Only show the locations for the codes selected
      a$ = aalist_code$[code_row]
      ; Search through the 10 ml array and find ones that match
      num_ten = 0
      startindex = 0
      sarrayfind startindex,acid_code$[startindex+1],a$
      while (startindex != -1 && startindex <= 100)
        num_ten = num_ten + 1
        code_index[num_ten] = startindex
        sarrayfind startindex,acid_code$[startindex+1],a$
      endwhile
      ; Search through the 2.5 ml array and find ones that match
      num_two = 0
      startindex = 100
      sarrayfind startindex,acid_code$[startindex+1],a$
      while (startindex != -1)
        code_index[num_two+101] = startindex - 100
        num_two = num_two + 1
        sarrayfind startindex,acid_code$[startindex+1],a$
      endwhile
    endif
    lastcoderow = code_row
    updatetable
    screenupdate
    enablekeys
  elseif (ten_row || two_row)
```

```
; Change amino acid
disablekeys
; Figure out which amino acid rack
if (ten_row)
  sel_row = code_index(ten_row)
else
  sel_row = code_index(two_row + 100) + 100
endif
show sel_row
ten_row = 0
two_row = 0
if (fast_fill)
  ; automatically set the syringe to the full amount
  if (acid_code$(sel_row) != "")
    acid_avail(sel_row) = 2.5 + 7.5*(sel_row<=100)
  endif
else
  ; allow the user to set any of the values
  if (sel_row <= 100)
    b$ = "\n(10 ml rack, position " + numtostr$(sel_row) + ")"
  else
    b$ = "\n(2.5 ml rack, position " + numtostr$(sel_row-100) + ")"
  endif
  ; input the code:  if changed, find the concentration from similar code
  a$ = acid_code$(sel_row)
  inputstring a$,"Enter the code:"+b$
  if (a$ != acid_code$(sel_row))
    acid_code$(sel_row) = a$
    sarrayfind startindex,acid_code$[1],a$
    if (startindex == sel_row)
      sarrayfind startindex,acid_code$[startindex+1],a$
    endif
    if (startindex != -1)
      acid_conc(sel_row) = acid_conc(startindex)
    else
      acid_conc(sel_row) = .1
    endif
    acid_avail(sel_row) = 0
endif
a = acid_conc(sel_row)
inputnum a,"Enter the concentration:"+b$
if (a != acid_conc(sel_row))
  startindex = 0
  sarrayfind startindex,acid_code$(startindex+1],a$
  while (startindex != -1)
    acid_conc(startindex) = a
    sarrayfind startindex,acid_code$(startindex+1],a$
  endwhile
endif
a = -1
while (a < 0 || a > (2.5 + 7.5*(sel_row<=100)))
  a = acid_avail(sel_row)
  inputnum a,"Enter the available volume:"+b$
  if (a >= 0 && a <= (2.5 + 7.5*(sel_row<=100)))
    if (a != acid_avail(sel_row))
```

```
          .cid_avail[sel_row] = a
        endif
      else
        tellalarm "The volume must be between 0 and " + numtostr$(2.5 + 7.5*(s
      endif
    endwhile
  endif
  updatetable
  screenupdate
  enablekeys
 endif
endwhile

savevars 1,"amino.dat"
savevars 2,"codes.dat"
screenback
```

```
;==========================================================
;    Procedure: BREAKSEQ
;    Purpose  : breaks sequence into individual codes
;    Author   : MJB
;    Notes    : Input  - bs_seq$
;               Output - couplings = # of couplings
;                        code$[couplings] = individual codes in sequence
;                        bs_error = 1 if error, 0 if valid sequence
;==========================================================

local char$,seqlen,pos,a,b

seqlen = strlen(bs_seq$)
pos = 1
couplings = 0
bs_error = 0
setstringarray code$[1],"",-1

// Break sequence string into peptide codes ==================
while (pos<=seqlen)
  char$ = mid$(bs_seq$,pos,1)
  if (isallalnum(char$))
    code$[couplings+1] = code$[couplings+1] + char$
  elseif (char$=="-")
    if (strlen(code$[couplings+1]))
      couplings = couplings + 1
    endif
  else
    bs_error = 1
    stop
  endif
  pos = pos + 1
endwhile
if (strlen(code$[couplings+1]))
  couplings = couplings + 1
endif

// Flip sequence around =====================================

a = 1

while (a<=couplings)
  flipcode$[a] = code$[a]
  a = a + 1
endwhile
```

```
b = couplings
a = 1

while (a<=couplings)
  code$[a] = flipcode$[b]
  a = a + 1
  b = b - 1
endwhile
```

```
;========================================================================
;        Procedure : CALC
;        Purpose   : Calculate required and available volumes
;        Author    : MJB,CES
;========================================================================

local syringe,curamino,curprocess,cursequence$,curdodep$,mytime

timer mytime
disablekeys
sysmsg$ = "Waiting for robot..."
screenupdate
stringrequest "USING SYRINGES"

sysmsg$ = "Calculating required volumes..."
calc_needed = 0
screenupdate

num_codes = 0
setstringarray aalist_code$[0],"",-1
setnumarray aalist_req[0],0,-1
setnumarray aalist_avail[0],0,-1
setnumarray aalist_conc[0],0,-1
setnumarray sol_req[1],0,-1
syringe = 1
while (syringe <= num_syringes)

  sysmsg$ = "Calculating volumes (syringe "+numtostr$(syringe)+")..."
  screenupdate
  ; Only process the syringe if it has been assigned a location or it
  ; has already been started
  if (syr_loc[syringe] || syr_process[syringe] > 1)
    ; Set the process data to the syringe's current status or to the
    ; start if it hasn't begun
    if (syr_process[syringe] > 1)
      curprocess = syr_process[syringe]
      curamino = syr_amino_step[syringe]
    else
      curprocess = 2
      curamino = 1
    endif
    cursequence$ = syr_sequence$[syringe]
    curdodep$ = syr_dodeprot$[syringe]
    aa_short = 0
    buf_short = 0

    ; Find the amino acid code
    bs_seq$ = syr_sequence$[syringe]
    call breakseq.tcl

    ; Go through the protocol and figure out how much amino acid and
    ; reagent is required at each process step
    while ((prot_proc[curprocess] != step_finished) && (curprocess < num_procs))

      show "Step:  " + prot_step$[prot_proc[curprocess]]
```

```
      ; Find amino volume required
      aa_process = curprocess
      aa_amino$ = code$[curamino]
      aa_syringe = syringe
      call calcaa.tcl

      ; Find solution and buffer volume
      buf_process = curprocess
      buf_syringe = syringe
      call calcbuf.tcl

      ; Determine which step is next
      stringrequest "NEXT STEP"
      ns_process = curprocess
      ns_amino = curamino
      ns_dodep$ = curdodep$
      call nextstep.tcl
      curprocess = ns_process
      curamino = ns_amino
      stringfree "NEXT STEP"

    endwhile

    ; Is there enough aa and buf for syringe to be processed?
    if (aa_short || buf_short)
      calc_needed = 1
      if (syr_process[syringe] == 1)
        syr_process[syringe] = 0
      elseif (syr_process[syringe] > 1)
        tellalarm "Syringe #"+numtostr$(syringe)+" is in process\nbut will not h
      endif
    elseif (syr_process[syringe] == 0)
      syr_process[syringe] = 1
    endif

  endif
  syringe = syringe + 1
endwhile

sysmsg$ = ""
updatetable
screenupdate
enablekeys

show "calc took " + numtostr$(elapsed(mytime))

stringfree "USING SYRINGES"
```

```
;-----------------------------------------------------------------------
;       Procedure : CALCAA
;       Purpose   : Calculate required amino acid for a step
;       Author    : MJB,CES
;       Notes     : Input: aa_process
;                          aa_amino$
;                          aa_syringe
;                          aa_short  -- set if the avail is less than required
;-----------------------------------------------------------------------

local c_index,rt_index,tempamount

if (prot_step$[prot_proc[aa_process]] == "Add Coupl.")

  ; Find the code in the "required" list, insert it if it isn't there
  sarrayfind c_index,aalist_code$[1],aa_amino$
  if (c_index == -1)
    num_codes = num_codes + 1
    aalist_code$[num_codes] = aa_amino$
    c_index = num_codes
    ; Now find the code in the RT racks and find the available volume
    sarrayfind rt_index,acid_code$[1],aa_amino$
    while (rt_index != -1)
      ; Find the total amount of amino acid
      aalist_avail[num_codes] = aalist_avail[num_codes] + acid_avail[rt_index]
      aalist_conc[num_codes] = acid_conc[rt_index]
      sarrayfind rt_index,acid_code$[rt_index+1],aa_amino$
    endwhile
    c_index = num_codes
  endif
  ; Determine the volume required
  if (aalist_conc[c_index])
    tempamount = syr_resin[aa_syringe] * syr_subst[aa_syringe] * prot_vol[aa_pro
    tempamount = tempamount / aalist_conc[c_index]
    aalist_req[c_index] = aalist_req[c_index] + tempamount
  else
    aalist_avail[c_index] = -1
    aalist_req[c_index] = 0
  endif

  if (aalist_req[c_index] > aalist_avail[c_index])
    aa_short = 1
  endif

endif
```

```
;========================================================================
;       Procedure  : CALCBUF
;       Purpose    : Calculate the required buffer solutions
;       Author     : MJB,CES
;       Notes      : Input: buf_process
;                           buf_syringe
;                           buf_short  -- set if avail is less than required
;========================================================================

local temp$,sol_index

temp$ = prot_step$[prot_proc(buf_process]]
if (temp$ != "Add Coupl." && temp$ != "Wash" && temp$ != "Add Deprot.")
  stop
endif

sarrayfind sol_index,contents$[1],prot_buf$[buf_process]
if (sol_index == -1)
  sysmsg$ = "Could not find buffer solution: "+prot_buf$[buf_process]
  delay 1000
  stop
endif

if (temp$ == "Wash")
  sol_req[sol_index] = sol_req[sol_index] + prot_vol[buf_process]
  sol_req[sol_index] = sol_req[sol_index] + cleaning_amount*3
elseif (temp$ == "Add Coupl.")
  sol_req[sol_index] = sol_req[sol_index] + (syr_resin[buf_syringe]*syr_subst[bu
elseif (temp$ == "Add Deprot.")
  sol_req[sol_index] = sol_req[sol_index] + prot_vol[buf_process]
endif

if (sol_req[sol_index] > sol_avail[sol_index])
  buf_short = 1
endif
```

```
;
;       Procedure: CLEANCUP
;       Purpose  : clean coupling cup with solution
;       Author   : MJB
;

local b

;find cleaning solution
sarrayfind clean_buf1,prot_step$[1],"Wash"
narrayfind clean_buf2,prot_proc[1],clean_buf1
sarrayfind clean_buf,contents$[1],prot_buf$[clean_buf2]

clean_cup_done = 1

b = 0
while (b<3)
  ;dispense cleaning solution
  if (!simulate_robot)
    SYRINGE.dispense clean_buf,cleaning_amount
  endif
  delay 500

  ;aspirate cleaning solution, fourth prob is vacuum probe
  if (!simulate_robot)
    SYRINGE.pipette 4,(cleaning_amount + .5)
  endif
  delay 500

  b = b + 1
endwhile

SYRINGE.pipette 4,cleaning_amount

clean_cup_done = 0
```

```
;================================================
;  Procedure: CLEARALL
;    Purpose: Clear all syringes from the system
;     Author: CES
;================================================

local choice

sysmsg$ = ""
screenupdate
clear_all_syringes = 0
ask choice,"Are you sure you want to clear all syringes?"
if (!choice)
  enablekeys
  stop
endif

if (next_syringe > 1)
  ask choice,"Are you sure you want to clear all active syringes?"
  if (!choice)
    enablekeys
    stop
  else
    tell "Please remove ALL SYRINGES from the ROBOT SYSTEM now."
  endif

  sysmsg$ = "Clearing all syringes"

endif

screenupdate
; data for a sample
setstringarray syr_pepcode$[1],"",-1
setstringarray syr_sequence$[1],"",-1
setnumarray syr_size[1],10,-1
setnumarray syr_rack[1],0,-1
setnumarray syr_rackloc[1],0,-1
setnumarray syr_loc[1],0,-1
setnumarray syr_grip[1],0,-1
setnumarray syr_amount[1],0,-1
setnumarray syr_process[1],0,-1
setnumarray syr_amino_step[1],0,-1        ;holds current coupling solution
setnumarray syr_amino_count[1],0,-1       ;holds current coupling solution
setnumarray syr_resin[1],.2,-1
setnumarray syr_subst[1],.45,-1
setstringarray syr_dodeprot$[1],"N",-1
setnumarray syr_processtime[1],0,-1
setstringarray syr_datetime$[1],"",-1
setnumarray syr_statcolor[1],0,-1
syr_pepcode$[1] = "Default"
num_syringes = 0
next_syringe = 1
current_row = 1
restart_index = 0
```

```
; Set required amino to zero
num_codes = 0
setstringarray aalist_code$[1],"",200
setnumarray aalist_req[1],0,200
setnumarray aalist_avail[1],0,200
setnumarray aalist_conc[1],0,200

screenupdate
savevars 5,"syringes.dat"
savevars 2,"codes.dat"
sysmsg$ = ""
screenupdate
enablekeys
```

```
;
;  Procedure: CLEARALL
;    Purpose: Clear all syringes from the system
;     Author: CES
;

local choice

sysmsg$ = ""
screenupdate
clear_all_syringes = 0
ask choice,"Are you sure you want to clear all syringes?"
if (!choice)
  enablekeys
  stop
endif

if (next_syringe > 1)
  ask choice,"Are you sure you want to clear all active syringes?"
  if (!choice)
    enablekeys
    stop
  else
    tell "Please remove ALL SYRINGES from the ROBOT SYSTEM now."
  endif

  sysmsg$ = "Clearing all syringes"

endif

sc1 .nupdate
; data for a sample
setstringarray syr_pepcode$[1],"",-1
setstringarray syr_sequence$[1],"",-1
setnumarray syr_size[1],10,-1
setnumarray syr_rack[1],0,-1
setnumarray syr_rackloc[1],0,-1
setnumarray syr_loc[1],0,-1
setnumarray syr_grip[1],0,-1
setnumarray syr_amount[1],0,-1
setnumarray syr_process[1],0,-1
setnumarray syr_amino_step[1],0,-1          ;holds current coupling solution
setnumarray syr_amino_count[1],0,-1         ;holds current coupling solution
setnumarray syr_resin[1],.2,-1
setnumarray syr_subst[1],.45,-1
setstringarray syr_dodeprot$[1],"N",-1
setnumarray syr_processtime[1],0,-1
setstringarray syr_datetime$[1],"",-1
setnumarray syr_statcolor[1],0,-1
syr_pepcode$[1] = "Default"
num_syringes = 0
next_syringe = 1
current_row = 1
restart_index = 0
```

```
; Set requi.. .. amino to zero
num_codes = 0
setstringarray aalist_code$[1],"",200
setnumarray aalist_req[1],0,200
setnumarray aalist_avail[1],0,200
setnumarray aalist_conc[1],0,200

screenupdate
savevars 5,"syringes.dat"
savevars 2,"codes.dat"
sysmsg$ = ""
screenupdate
enablekeys
```

```
;------------------------------------------------------------
;       Procedure: CLRRACK
;       Purpose  : places a zero in rack position
;                  and current position in gripper
;       Author   : MJB
;------------------------------------------------------------

;clear input rack
if (syr_loc[current_index])
  syr_loc[current_index] = 0
endif

;set rack to gripper
syr_rack[current_index] = 10

;clear rack location
syr_rackloc[current_index] = 0

;save changed rack varibles
savevars 5,"syringes.dat"
```

```
;===================================================================
;       Procedure: COUPLNG
;       Purpose  : robot actions for coupling
;       Author   : MJB
;===================================================================

local b,buf_place,infoindex

infoindex = 1

;find buffer in reagent table
sarrayfind buf_place,contents$[1],prot_buf$[syr_process[current_index]]

;put RX syringe in holding rack if RX syringe in gripper
if (syr_rack[current_index]==10)
  if (syr_size[current_index]==10)
    rack_type = 1
  else
    rack_type = 2
  endif
  nest = 1
  grip = syr_grip[current_index]
  call puthold.tcl
  call stopchek.tcl
endif

;get RT syringe and dispense amino acid
getting_rt = 1
show cp_amino$
sar  rfind a,acid_code$[1],cp_amino$
aciu_needed = syr_resin[current_index]*syr_subst[current_index]*prot_vol[syr_pro
amino_disp = acid_needed
amino_found = 0
amino_place = 0
while (!amino_found)
  sarrayfind amino_place,acid_code$[(amino_place+1)],cp_amino$
  if (acid_avail[(amino_place)])
    amino_found = 1
  endif
endwhile
while ((acid_avail[amino_place]+.001)<acid_needed)   ;get as many amino syringes
  rack_location = amino_place
  if (amino_place>100)
    rack_location = rack_location - 100
    rack_type = 4
  else
    rack_type = 3
  endif
  call getrack.tcl
  call stopchek.tcl
  grip = 1
  if (amino_place>100)
    syringe = 2
  else
    ~yringe = 1
```

```
 endif
 nest = 2
 call puthold.tcl
 call stopchek.tcl
 grip = 2
 nest = 2
 call gethold.tcl
 call stopchek.tcl
 cup = 1
 restart_info$[infoindex] = "Add Coup:  Began Dispensing " + numtostr$(dsp_amou
 savevars 7,"restart.dat"
 infoindex = infoindex + 1
 dsp_amount = acid_avail[amino_place]
 call syr_dsp.tcl
 call stopchek.tcl
 restart_info$[infoindex] = "Add Coup:  Finished Dispensing " + numtostr$(dsp_a
 savevars 7,"restart.dat"
 infoindex = infoindex + 1
 acid_needed = acid_needed - acid_avail[amino_place]
 acid_avail[amino_place] = 0
 savevars 1,"amino.dat"
 grip = 2
 nest = 2
 call puthold.tcl
 call stopchek.tcl
 grip = 1
 nest = 2
 call gethold.tcl
 call stopchek.tcl
 rack_location = amino_place
 if (amino_place>100)
  rack_location = rack_location - 100
  rack_type = 4
 else
  rack_type = 3
 endif
 call putrack.tcl
 call stopchek.tcl
 amino_found = 0
 while (!amino_found)
   sarrayfind amino_place,acid_code$[(amino_place+1)],cp_amino$
   if ((acid_avail[amino_place])||(amino_place==-1))
     amino_found = 1
   endif
 endwhile
endwhile
rack_location = amino_place     ;do final RT syringe dispense
if (amino_place>100)
  rack_location = rack_location - 100
  rack_type = 4
else
  rack_type = 3
endif
call getrack.tcl
call stopchek.tcl
```

-46-

```
grip = 1
if (amino_place>100)
  syringe = 2
else
  s inge = 1
endif
nest = 2
call puthold.tcl
call stopchek.tcl
grip = 2
nest = 2
call gethold.tcl
first_pulses = motor_pulses
call stopchek.tcl
cup = 1
dsp_amount = acid_needed
restart_info$[infoindex] = "Add Coup:  Began Dispensing " + numtostr$(dsp_amount
savevars 7,"restart.dat"
infoindex = infoindex + 1
call syr_dsp.tcl
call stopchek.tcl
acid_avail[amino_place] = acid_avail[amino_place] - acid_needed
savevars 1,"amino.dat"
restart_info$[infoindex] = "Add Coup:  Finished Dispensing " + numtostr$(dsp_amo
savevars 7,"restart.dat"
infoindex = infoindex + 1
grip = 2
nest = 2
check_motor = 1
cal  puthold.tcl
che  _motor = 0
call stopchek.tcl
second_pulses = motor_pulses

;check that fluid was dispensed from syringe
if (!simulate_robot)
  if (  ((first_pulses-5)<second_pulses) && (second_pulses<(first_pulses+5)) )
    error_code = 3
    call error.tcl
  endif
endif

grip = 1
nest = 2
call gethold.tcl

call stopchek.tcl
rack_location = amino_place
if (amino_place>100)
  rack_location = rack_location - 100
  rack_type = 4
else
  rack_type = 3
endif
ca]' putrack.tcl
```

```
call stopchek.tcl
getting_rt = 0

;dispense buffer solution
restart_info$[infoindex] = "Add Coup:  Began Dispensing Buffer"
savevars 7,"restart.dat"
infoindex = infoindex + 1
cup = 1
sarrayfind a,contents$[1],prot_buf$[syr_process[current_index]]
disp_amount = syr_resin[current_index]*syr_subst[current_index]*prot_vol[syr_pro
reagent_disp = disp_amount
buf_number = a
buf_amount = disp_amount
run disp_buf.tcl
sol_avail[a] = sol_avail[a] - disp_amount
savevars 4,"reagents.dat"
if (log_robot)
  logrobot$ = "dispense : " + numtostr$(disp_amount) + " nozzle: " + numtostr$(a
  call logrobot.tcl
  logrobot$ = "solution : " + prot_buf$[syr_process[current_index]]
  call logrobot.tcl
endif
restart_info$[infoindex] = "Add Coup:  Finished Dispensing Buffer"
savevars 7,"restart.dat"
infoindex = infoindex + 1

;get RX syringe from holding rack
if (syr_size[current_index]==10)
  syringe = 1
else
  syringe = 2
endif
grip = 2
nest = 1
call gethold.tcl
syr_grip[current_index] = 2
call stopchek.tcl

;squeeze out solution in syringe
if ((syr_size[current_index]==10&&syr_amount[current_index]>drop_amount1)||(syr_
  call waste.tcl
  call stopchek.tcl
endif
restart_info$[infoindex] = "Add Coup:  Emptied syringe"
savevars 7,"restart.dat"
infoindex = infoindex + 1

;aspirate solution in cup
restart_info$[infoindex] = "Add Coup:  Began Aspirating Solution"
savevars 7,"restart.dat"
infoindex = infoindex + 1
asp_amount = disp_amount + amino_disp
air_amount = prot_air[syr_process[current_index]]
call syr_asp.tcl
call stopchek.tcl
```

```
restart_info$[infoindex] = "Add Coup:  Finished Aspirating Solution"
savevars 7,"restart.dat"
infoindex = infoindex + 1

;c  n cup
if (!simulate_robot)
  run cleancup.tcl
endif
```

```
;----------------------------------------------------------------
;       Procedure: DATRACK
;       Purpose  : places a gripper value in rack position
;       Author   : MJB
;----------------------------------------------------------------

;set syringe rack for current index
syr_rack[current_index] = rack_type

;clear rack location
syr_rackloc[current_index] = rack_location

;save changed rack variables
savevars 5,"syringes.dat"
```

-50-

```
;--------------------------------------------------------
;  Procedure: DELETE
;    Purpose: Delete a syringe from the system
;     Author: CES
;--------------------------------------------------------

local choice,temp,a$,size$

if (prot_proc(syr_process(delete_row)] != step_finished && current_index == dele
  tellalarm "You must wait until the syringe is in the tumbler!"
  stop
endif

if (syr_process(delete_row) > 1 && prot_proc(syr_process(delete_row)] != step_fi
  ask choice,"The peptide is in progress.\nAre you sure you want to delete pepti
  if (!choice)
    stop
  endif
  sysmsg$ = "Waiting for robot..."
  screenupdate
  stringrequest "USING SYRINGES"
  syr_process(delete_row) = num_procs
  syr_processtime(delete_row) = time_now
  sysmsg$ = ""
  screenupdate
  tell "The peptide has been marked as finished.\nThe robot will eventually take
  stringfree "USING SYRINGES"
else
  if (user_choice == 3)

    if (prot_proc(syr_process(delete_row)] == step_finished)
      tell "The robot will remove this syringe automatically.\nPlease allow ROBO
      stop
    else
      ask choice,"Are you sure you want to delete peptide, " + syr_pepcode$(dele
    endif

    if (!choice)
      stop
    else
      size$ = "from " + numtostr$(syr_size(delete_row)) + "ml INPUT RACK locatio
      tell "Please physically REMOVE this syringe\n" + size$ + numtostr$(syr_loc
    endif

  endif

; Write the information to the complete.dat file
if (prot_proc(syr_process(delete_row)] == step_finished)
  stringrequest "SYRFILE"
  fileopen 6,"complete.dat"
  sf_row = delete_row
  call syrfile.tcl
  fileclose 6
  stringfree "SYRFILE"
endif
```

```
  sysmsg$ = "Deleting from list..."
  temp = delete_row
  while (temp <= num_syringes)
    syr_pepcode$(temp] = syr_pepcode$[temp+1]
    syr_size(temp] = syr_size(temp+1]
    syr_loc(temp] = syr_loc(temp+1]
    syr_sequence$[temp] = syr_sequence$[temp+1]
    syr_process(temp] = syr_process(temp+1]
    syr_amino_step(temp] = syr_amino_step(temp+1]
    syr_amino_count(temp] = syr_amino_count(temp+1]
    syr_resin(temp] = syr_resin(temp+1]
    syr_subst(temp] = syr_subst(temp+1]
    syr_dodeprot$(temp] = syr_dodeprot$(temp+1]
    syr_processtime(temp] = syr_processtime[temp+1]
    syr_datetime$[temp] = syr_datetime$(temp+1]
    syr_statcolor(temp] = syr_statcolor(temp+1]
    syr_amount(temp] = syr_amount(temp+1]
    syr_grip(temp] = syr_grip(temp+1]
    syr_rack(temp] = syr_rack(temp+1]
    syr_rackloc(temp] = syr_rackloc(temp+1]
    temp = temp + 1
  endwhile
  num_syringes = num_syringes - 1
  if (next_syringe > delete_row)
    next_syringe = next_syringe - 1
  endif

endif

sysmsg$ = "Saving data to file..."
screenupdate
if (num_syringes == 0)
  ; Set required amino to zero
  num_codes = 0
  setstringarray aalist_code$[1],"",200
  setnumarray aalist_req(1),0,200
  setnumarray aalist_avail(1),0,200
  setnumarray aalist_conc(1),0,200
  savevars 2,"codes.dat"
endif
savevars 5,"syringes.dat"
sysmsg$ = ""
screenupdate
```

-52-

```
;============================================================
;        Procedure: DEPROTCT
;        Purpose  : robot actions for a deprotection
;        Author   : MJB
;============================================================

local buf_place,infoindex

infoindex = 1

;find buffer in reagent table
sarrayfind buf_place,contents$[1],prot_buf$[syr_process[current_index]]

;change to stroke grip
if (syr_grip[current_index]==1||(syr_rack[current_index]==8||syr_rack[current_i
  if (syr_size[current_index]==10)
    syringe = 1
  else
    syringe = 2
  endif
  grip = 1
  nest = 1
  if (syr_grip[current_index]==1)
    call puthold.tcl
    call stopchek.tcl
  endif
  if (syr_size[current_index]==10)
    syringe = 1
  else
    yringe = 2
  e. .if
  grip = 2
  nest = 1
  call gethold.tcl
  syr_grip[current_index] = 2
  call stopchek.tcl
endif

;squeeze out solution in syringe
if ((syr_size[current_index]==10&&syr_amount[current_index]>drop_amount1)||(syr_
  call waste.tcl
  call stopchek.tcl
endif

restart_info$[infoindex] = "Add Deprot:  Emptied syringe"
savevars 7,"restart.dat"
infoindex = infoindex + 1

;dispense buffer
restart_info$[infoindex] = "Add Deprot:  Started dispensing buffer"
savevars 7,"restart.dat"
infoindex = infoindex + 1
buf_number = buf_place
buf_amount = prot_vol[syr_process[current_index]]
.f 7<yringe==2)
```

-53-

```
  buf_amount = buf_amount/division_factor
endif
run disp_buf.tcl
sol_avail[buf_place] = sol_avail[buf_place] - buf_amount
savevars 1,"amino.dat"
if (log_robot)
  logrobot$ = "dispense : " + numtostr$(buf_amount) + " nozzle: " + numtostr$(bu
  call logrobot.tcl
  logrobot$ = "solution : " + prot_buf$[syr_process[current_index]]
  call logrobot.tcl
endif
restart_info$[infoindex] = "Add Deprot:  Finished dispensing buffer"
savevars 7,"restart.dat"
infoindex = infoindex + 1

;aspirate buffer
restart_info$[infoindex] = "Add Deprot:  Started aspirating buffer"
savevars 7,"restart.dat"
infoindex = infoindex + 1
cup = 2
if (syr_size[current_index]==10)
  syringe = 1
else
  syringe = 2
endif
asp_amount = buf_amount
air_amount = prot_air[syr_process[current_index]]
if (syringe==2)
  air_amount = air_amount/division_factor
endif
call syr_asp.tcl
call stopchek.tcl
restart_info$[infoindex] = "Add Deprot:  Finished aspirating buffer"
savevars 7,"restart.dat"
infoindex = infoindex + 1

;rotate
rot_time = 60*prot_time[syr_process[current_index]]
call rotate.tcl
call stopchek.tcl
```

```
;
;  Procedure: DOSTEP
;    Purpose: Perform the next step for the syringe
;     Author: CES
;      Notes: current_index is the syringe to work on
;

local process

process = prot_proc(syr_process(current_index))

if (process == step_ready)
  syr_rack(current_index) = (2-(syr_size(current_index)-2.5)/7.5)
  syr_rackloc(current_index) = syr_loc(current_index)
  syr_loc(current_index) = 0
  syr_amino_step(current_index) = 1

  ; This should always be true if we got here
  if (current_index == next_syringe)
    next_syringe = next_syringe + 1
  else
    tellalarm "Logic error!  next_syringe updated incorrectly"
  endif
  updatetable

elseif (process == step_wash)
  call washing.tcl
  logrobot$ = prot_step$(process)
  if (log_procedure)
    call logrobot.tcl
  endif
  if (dont_wait)
    time_now = time_now + prot_robottime(syr_process(current_index))
  else
    time_now = elapsed(0)
  endif

elseif (process == step_adddeprot)
  call deprotct.tcl
  logrobot$ = prot_step$(process)
  if (log_procedure)
    call logrobot.tcl
  endif
  if (dont_wait)
    time_now = time_now + prot_robottime(syr_process(current_index))
  else
    time_now = elapsed(0)
  endif

elseif (process == step_addcoupl)
  bs_seq$ = syr_sequence$(current_index)
  call breakseq.tcl
  cp_amino$ = code$(syr_amino_step(current_index))
  call coupling.tcl
  logrobot$ = "Add Coupling: " + code$(syr_amino_step(current_index))
```

```
    if (log_procedure)
     call logrobot.tcl
    endif
    if (dont_wait)
      time_now = time_now + prot_robottime[syr_process[current_index]]
    else
      time_now = elapsed(0)
    endif

 elseif (process == step_deprotection || process == step_coupling)
   if (syr_rack[current_index] != 5 && syr_rack[current_index] != 6)
     call put_tum.tcl
     logrobot$ = "Put Tumbler\n"
     if (log_procedure)
       call logrobot.tcl
     endif
     if (dont_wait)
       time_now = time_now + prot_robottime[syr_process[current_index]]
     else
       time_now = elapsed(0)
     endif
     syr_processtime[current_index] = time_now + 60*prot_time[syr_process[current
     current_index = 0
   else
     call get_tum.tcl
     logrobot$ = "Get Tumbler, overtime -- "+numtostr$(time_now-syr_processtime[c
     if (log_procedure)
       call logrobot.tcl
     endif
     if (dont_wait)
       time_now = time_now + prot_robottime[syr_process[current_index]]
     else
       time_now = elapsed(0)
     endif
   endif

elseif (process == step_finished)
  ;dump off syringe in waste basket
  logrobot$ = prot_step$[process]
  if (log_procedure)
    call logrobot.tcl
  endif
  call finish.tcl
  ;delete syringe from list
  delete_row = current_index
  stringfree "USING SYRINGES"
  call delete.tcl
  stringrequest "USING SYRINGES"
  current_index = 0
endif
```

```
;===================================================================
;       Procedure: DUMP
;       Purpose  : drop finished syringe in basket
;       Author   : MJB
;===================================================================

if (log_robot)
  logrobot$ = "dump -- " + numtostr$(current_index)
  call logrobot.tcl
endif

if (simulate_robot)
  stop
endif

request "ROBOT"

ROBOT.speed sp_fast

ROBOT.move "drop"

ROBOT.speed sp_slow

ROBOT.jog_s 0,0,-drop_jog
ROBOT.finish

ROBOT.open
delay 1000

ROBOT.jog_s 0,0,drop_jog

ROBOT.speed sp_med

free
```

```
;----------------------------------------------------------------
;  Procedure: EQUPINIT
;    Purpose: Starting procedure for robot and Hamilton
;     Author: MJB
;      Notes:
;----------------------------------------------------------------

disablekeys
initialization = 0
sysinit_message$ = "Please Wait..."
robot_ready = 0    ; set a bit for each question
syringe_ready = 0
screenon "SYSINIT"
if (!simulate_robot)
  call robhome.tcl
else
  robot_ready = 1
endif
if (!robot_ready)
  tell "Initialization cancelled"
  free
  screenback
  stop
endif

;/* Prepare Syringe Pump */

if (!simulate_robot)
  call syrinit.tcl
else
  syringe_ready = 1
endif
if (!syringe_ready)
  tell "Initialization cancelled"
  free
  screenback
  stop
endif

;/* Homing Motor */
sysinit_message$ = "Homing Motor"
if(!simulate_robot)
  call homemtr.tcl
  call safemtr.tcl
endif

sysinit_message$ = ""
screenupdate
tell "Initialization complete"
initialization = 1
screenback
```

```
;------------------------------------------------------------
;       Procedure: FINISH
;       Purpose  : dumps off finished syringe
;       Author   : MJB
;------------------------------------------------------------

if (syr_rack(current_index) == 5 || syr_rack(current_index) == 6)
  call get_tum.tcl
endif

;if in gripper, check grip, change if necessary
if (syr_rack(current_index)==10)
  if (syr_grip(current_index)==2)
    nest = 1
    grip = 2
    if (syr_size(current_index)==10)
      syringe = 1
    else
      syringe = 2
    endif
    call puthold.tcl
    syr_rack(current_index) = 0
    call stopchek.tcl
    nest = 1
    grip = 1
    if (syr_size(current_index)==10)
      syringe = 1
    else
      syringe = 2
    ndif
    call gethold.tcl
    syr_grip(current_index) = 1
    syr_rack(current_index) = 10
    call stopchek.tcl
  endif

;if in holding rack, get with move grip
elseif (syr_rack(current_index)==8||syr_rack(current_index)==9)
  if (syr_size(current_index)==10)
    syringe = 1
  else
    syringe = 2
  endif
  grip = 1
  nest = 1
  call gethold.tcl
  syr_grip(current_index) = 1
  syr_rack(current_index) = 10
  call stopchek.tcl

;if in other rack, get with move grip
else
  if (syr_size(current_index)==10)
    syringe = 1
  e'se
```

```
      syringe = 2
    endif
    call getrack.tcl
    syr_grip[current_index] = 1
    syr_rack[current_index] = 10
    call stopchek.tcl

  endif

  ;go to dump

  call dump.tcl
```

```
;---------------------------------------------------------------
; Procedure: GETHOLD
;   Purpose: Get a syringe from the holding nest
;    Author: MJB
;     Notes:
;---------------------------------------------------------------

;---------------------------------------------------------------
;         set syringe = 1 for 10ml , 2 for 2.5ml
;             grip    = 1 for move grip , 2 for stroke grip
;             nest    = 1 for nest 1, 2 for nest 2
;---------------------------------------------------------------
if (syringe==1)
  rack_location = 1
  rack_type = 8
else
  rack_location = 1
  rack_type = 9
endif

local get_repeat

get_repeat = 0

if (log_robot)
  logrobot$ = "gethold -- syringe :" + numtostr$(syringe) + " grip : " + numtost
  call logrobot.tcl
endif

if (.simulate_robot)
  if (!getting_rt)
    call clrrack.tcl
  endif
  stop
endif

request "ROBOT"

syringe_present = 1

while ((!syringe_present)||(!get_repeat))

;  if (get_repeat)
;    call puthold.tcl
;  endif

  ROBOT.open
  delay 500

  ROBOT.speed sp_fast

  if (grip!=2)
;    ROBOT.move "mixsafe1"
  else
```

```
;    ROBOT.move "mixsafe2"
  run homamtr.tcl
 endif

 a$ = "tmp" + numtostr$(syringe) + "_" + numtostr$(grip) + "_" + numtostr$(n

 ROBOT.speed sp_med

 ROBOT.move a$
 ROBOT.finish

 if ((!syringe_present)&&(get_repeat))
   error_code = 1        /** CHANGE THIS CODE IF NECESSARY **/
   call error.tcl
 endif

 ROBOT.speed sp_slow

 if (grip==2)
   while (!home_done)
   endwhile
 endif

 a$ = "tlv" + numtostr$(syringe) + "_" + numtostr$(grip) + "_" + numtostr$(nest

 if (grip==2)
   ROBOT.speed sp_mslow
 endif

 ROBOT.move_s a$
 ROBOT.finish

 if (grip==2)

   /* Routine to find the top of the syringe*/
   call findtop.tcl
   if (syringe==1)
     direction = 0
     motor_steps = syrg1_back
   endif
   if (syringe==2)
     direction = 0
     motor_steps = syrg2_back
   endif
   call strkmtr.tcl
   /*--------------------------------------------*/
endif

delay 1000
ROBOT.close
delay 500

if (grip==1)
  ROBOT.speed sp_slow
```

```
  else
    ROBOT.speed sp_vslow
  endif

 ;  = "tmp" + numtostr$(syringe) + "_" + numtostr$(grip) + "_" + numtostr$(nest
  ROBOT.move_s a$
  ROBOT.finish

;. a$ = "mixsafe" + numtostr$(grip)

  ROBOT.speed sp_med

;  ROBOT.move a$

  call syr_chk.tcl          ;find syringe

  get_repeat = get_repeat + 1

endwhile

if (!getting_rt)
  syr_grip[current_index] = grip
  call clrrack.tcl
endif

free
```

```
;
  GETRACK
; pick up bottle from any rack and return to safe position
;

/***************************
USE OF PROCEDURE:

before calling this procedure set the variable:
    rack_type = the type of rack
    1 - 10ml RX
    2 - 2.5ml RX
    3 - 10ml RT
    4 - 2.5ml RT
    5 - 10ml TUM
    6 - 2.5ml TUM
    7 - probe
    rack_location = the location in the rack

*************************************/

grip = 1

if (log_robot)
  logrobot$ = "getrack -- rack: " + numtostr$(rack_type) + " loc: " + numtostr$(
  call logrobot.tcl
endif

if (simulate_robot)
  if (!getting_rt)
    call clrrack.tcl
  endif
  stop
endif

request "ROBOT"

ROBOT.speed sp_vfast

if (rack_type==5)
  tum1_continue=0
  while (!tum1_latched)
  endwhile
elseif (rack_type==6)
  tum2_continue=0
  while (!tum2_latched)
  endwhile
endif

ROBOT.open

found = 0
robot_i = 0
while (!found)
  if (rack_location <= robot_i * row_size(rack_type))
```

```
      if ((rack_type==3)||(rack_type==4))
        ROBOT.move "rack_im"
      endif

      rack_temp$ = rack_high$(rack_type) + numtostr$(robot_i)
      ROBOT.move rack_temp$
      ROBOT.finish
      found = 1
    else
      robot_i = robot_i + 1
    endif
  endwhile

  ROBOT.speed sp_fast

  jog_temp = (track_jog(rack_type)*((row_size(rack_type)*robot_i) - rack_location)
  ROBOT.joint 6,-jog_temp
  ROBOT.finish

  ROBOT.speed sp_slow
  ROBOT.jog_s 0,0,-rack_jog(rack_type)
  ROBOT.finish
  delay 500
  ROBOT.close
  delay 500

  ROBOT.speed sp_slow
  ROBOT.jog_s 0,0,rack_jog(rack_type)
  ROBOT.finish

  if (rack_type!=7)
    ROBOT.speed sp_fast
    call syr_chk.tcl            ;find syringe
    while (!syringe_present)
      ROBOT.speed sp_med
      ROBOT.move "mixsafe1"
      ROBOT.move rack_temp$
      ROBOT.joint 6,-jog_temp
      error_code = 1          /** CHANGE THIS CODE IF NECESSARY **/
      call error.tcl
      ROBOT.open
      delay 200
      ROBOT.finish
      ROBOT.speed sp_slow
      ROBOT.jog_s 0,0,-rack_jog(rack_type)
      ROBOT.finish
      ROBOT.close
      delay 200
      ROBOT.jog_s 0,0,rack_jog(rack_type)
      call syr_chk.tcl
    endwhile
  endif

  ROBOT.speed sp_fast
```

```
if (!getting_rt)
  syr_grip(current_index) = 1
  c•ll clrrack.tcl
en

if(rack_type==5)
  tum1_continue=1
elseif (rack_type==6)
  tum2_continue=1
endif

free
```

```
;====================================================================
; Procedure: GET_TUM
;   Purpose: get syringe from tumbler
;    Author: MJB
;====================================================================

rack_type = syr_rack(current_index)
rack_location = syr_rackloc(current_index)

call getrack.tcl
syr_grip(current_index) = 1
call stopchek.tcl
```

```
;==================================================
;       Procedure: HOMEMTR
;       Purpose  : turn on motor until top sensor is triggered
;       Author   : MJB
;==================================================

home_done = 0

;turn on motor for backward direction

SHARK.writebit 224,0
delay 500
SHARK.writebit 404,1

SHARK.readbit i_sense_home,a

tester1 = 0

timer homemtr_time
while (!a)
  SHARK.readbit i_sense_home,a
  if (elapsed(homemtr_time)>30)

    ;stop motor from homing
    SHARK.writebit 404,0

    ;call error routine
    error_code = 4
    call error.tcl

    ;stroke motor down
    direction = 1
    motor_steps = 20
    call strkmtr.tcl

    ;start motor to home again
    SHARK.writebit 224,0
    delay 500
    SHARK.writebit 404,1
    timer homemtr_time

  endif
endwhile

home_done = 1
```

```
;==============================================================
;        Procedure: INIT
;        Purpose  : initialize system variables
;        Author   : MJB
;==============================================================

; Protocol procedure descriptions and parameter mask
param_buf = 1
param_vol = 2
param_air = 4
param_time = 8
param_condition = 16
param_robottime = 32
param_stop = 64
setstringarray prot_step$[0],"",-1
setnumarray prot_choice[0],0,-1
step_notready = 0
step_ready = 1
step_wash = 2
step_adddeprot = 3
step_addcoupl = 4
step_deprotection = 5
step_coupling = 6
step_nextamino = 7
step_stopifnofinaldep = 8
step_finished = 9
prot_step$[step_notready] = "Not Ready"
prot_step$[step_ready] = "Ready"
prot_step$[step_wash] = "Wash"
prot_choice[step_wash] = param_buf + param_vol + param_air +
param_time + param_robottime
prot_step$[step_adddeprot] = "Add Deprot."
prot_choice[step_adddeprot] = param_buf + param_vol + param_air +
param_time + param_robottime
prot_step$[step_addcoupl] = "Add Coupl."
prot_choice[step_addcoupl] = param_buf + param_vol + param_air +
param_robottime
prot_step$[step_deprotection] = "Deprotection"
prot_choice[step_deprotection] = param_time + param_stop
prot_step$[step_coupling] = "Coupling"
prot_choice[step_coupling] = param_time + param_stop
prot_step$[step_nextamino] = "Next Amino"
prot_step$[step_stopifnofinaldep] = "Stop If No Final Dep."
prot_choice[step_stopifnofinaldep] = param_condition
prot_step$[step_finished] = "Finished"
prot_choice[step_finished] = param_stop

; Variables to keep track of syringes in system
num_syringes = 0     ; total number of syringes that have been input
next_syringe = 1     ; next syringe that will be removed from the
incoming rack
getting_rt = 0        ;flag to tell an RT syringe is in system
dont_do_next = 0      ; flag to tell max number of syringes in
system
```

```
clear_all_syringes = 0 ; flag to tell robot procedure to clear all
syringes
setstringarray restart_info$[1],"",-1

; data for a sample
setstringarray syr_pepcode$[1],"",-1
setstringarray syr_sequence$[1],"",-1
setnumarray syr_size[1],10,-1
setnumarray syr_rack[1],0,-1
setnumarray syr_rackloc[1],0,-1
setnumarray syr_loc[1],0,-1
setnumarray syr_grip[1],0,-1
setnumarray syr_amount[1],0,-1
setnumarray syr_process[1],0,-1
setnumarray syr_amino_step[1],0,-1         ;holds current coupling
solution
setnumarray syr_amino_count[1],0,-1        ;holds current coupling
solution
setnumarray syr_resin[1],.2,-1
setnumarray syr_subst[1],.45,-1
setstringarray syr_dodeprot$[1],"N",-1
setnumarray syr_processtime[1],0,-1
setnumarray syr_statcolor[1],0,-1
setstringarray syr_datetime$[1],"",-1
syr_pepcode$[1] = "Default"

;// Record structure for amino acids, set once then retrieve from
disk
setstringarray acid_code$[1],"",200
setnumarray acid_conc[1],0,200
setnumarray acid_avail[1],0,200
fast_fill = 0

  Number of amino acids that appear in sequences
num_codes = 0

; Number of steps in current protocol
num_procs = 0

;Robot Speed Variables
sp_vslow = 5
sp_mslow = 20
sp_slow = 30
sp_mod = 50
sp_med = 70
sp_fast = 90
sp_vfast = 100

; Rack Variables
rack5_full = 0 ;set to keep track of whether or not tumblers are
full
rack6_full = 0
rt_get = 0      ;set to RT rack location when RT in system
rack_type = 1  ; 1 = 10ml RX syringe
```

```
                    ; 2 = 2.5ml RX syringe
                    ; 3 = 10ml RT syringe
                    ; 4 = 2.5ml RT syringe
                    ; 5 = 10ml tumbler
                    ; 6 = 2.5ml tumbler
                    ; 7 = probe
                    ; 8 = 10ml holding
                    ; 9 = 2.5ml holding
                    ; 10 = gripper
row_size[1] = 6
row_size[2] = 6
row_size[3] = 20
row_size[4] = 20
row_size[5] = 8
row_size[6] = 8
row_size[7] = 5
rack_jog[1] = 8
rack_jog[2] = 8
rack_jog[3] = 8
rack_jog[4] = 8
rack_jog[5] = 8
rack_jog[6] = 8
rack_jog[7] = 4
track_jog[1] = 1.5
track_jog[2] = 1.5
track_jog[3] = 1.5
track_jog[4] = 1.5
track_jog[5] = 1.5
track_jog[6] = 1.5
track_jog[7] = 2
rack_safe$[1] = "RXSAFE"
rack_safe$[2] = "RXSAFE"
rack_safe$[3] = "RTSAFE"
rack_safe$[4] = "RTSAFE"
rack_safe$[5] = "TUMSAFE"
rack_safe$[6] = "TUMSAFE"
rack_safe$[7] = "MIXSAFE1"
rack_high$[1] = "RX1_"
rack_high$[2] = "RX2_"
rack_high$[3] = "RT1_"
rack_high$[4] = "RT2_"
rack_high$[5] = "TUM1_"
rack_high$[6] = "TUM2_"
rack_high$[7] = "PRB_"

; / Syringe Variables /
syringe = 1        ; 1 is 10ml syringe, 2 is 2.5ml syringe
temp_jog = 8

; / Cup Variables /
cup = 1     ;1 for RGP1, 2 for RGP2
cup_jog = 4

; / Waste Variables /
```

```
waste_jog = 3

; / Finished Peptide Basket Variables /
drop_jog = 8

; / Check Syringe Presence Variables /
syringe_sense_im$[1] =       "IM_1"
syringe_sense_im$[2] =       "IM_2"
syringe_sense_begin1$[1] =   "CKB_11"
syringe_sense_begin1$[2] =   "CKB_12"
syringe_sense_end1$[1] =     "CKE_11"
syringe_sense_end1$[2] =     "CKE_12"
syringe_sense_begin2$[1] =   "CKB_21"
syringe_sense_begin2$[2] =   "CKB_22"
syringe_sense_end2$[1] =     "CKE_21"
syringe_sense_end2$[2] =     "CKE_22"
syringe_sense_im$[3] = "IM_3"
syringe_sense_begin1$[3] = "CKB_13"
syringe_sense_end1$[3] = "CKE_13"
syringe_sense_begin2$[3] = "CKB_23"
syringe_sense_end2$[3] = "CKE_23"

; / Shark Inputs /

s = 24
a = 0

;while (s==24||!a)

  erroroff
  table_position = 1
  system_start = 0
  sense_table = 1
  sense_syringe = 2
  sense_press = 3
  sense_home = 4
  sensor_latch = 551
  motor_finish = 225
  sense_tum1 = 5
  sense_tum2 = 6
  SHARK.puttable
table_position,system_start,sense_table,sense_syringe,sense_press,s
ense_home,sensor_latch,motor_finish,sense_tum1,sense_tum2
  i_system_start = 1
  i_sense_table = 2
  i_sense_syringe = 3
  i_sense_press = 4
  i_sense_home = 5
  i_sensor_latch = 6
  i_motor_finish = 7
  i_sense_tum1 = 8
  i_sense_tum2 = 9
  SHARK.readbit i_system_start,a
  status s
```

```
;counter inputs
pulses_counter = 2
putelapsed 10,pulses_counter
i_motor_pulses = 10

;  if (s==24||!a)
;     tell "Turn the Selector Switch to ON\nand pull out the
Emergency Stop button"
;  endif
;endwhile

erroron

; / Shark Outputs /
o_tum1_start = 220
o_tum2_start = 221
o_tum1_clamp = 222
o_tum1_latch = 223
o_grip_dir = 224
o_grip_pulse = 225
o_buzzer = 226
o_syringe_locate = 227
o_tum2_clamp = 240
o_tum2_latch = 241
o_start_counter = 414
o_clear_counter = 415
o_clear_sensor = 550

; / Initialize Shark Outputs/
SHARK.writebit o_tum1_start,0       ;up
SHARK.writebit o_tum2_start,0       ;up
SHARK.writebit o_tum1_latch,0    ;open
SHARK.writebit o_tum2_latch,0   ;off
SHARK.writebit o_tum1_clamp,1    ;open
SHARK.writebit o_tum2_clamp,1   ;off
SHARK.writebit o_grip_dir,0       ;off
SHARK.writebit o_grip_pulse,0     ;open
SHARK.writebit o_buzzer,0       ;off

; / Error Variables /
error = 0
error_code = 0
err_syringe_sensor = 2
err_syringe_presence = 3

; / Error Messages /
error_message$[0] = ""
error_message$[100] = ""
error_message$[1] = "No Syringe Found"
error_message$[2] = "Syringe Sensor Not Reset"
error_message$[3] = "Syringe Did Not Dispense"
error_message$[4] = "Motor Did Not Home"
error_message$[5] = "Top of Syringe Not Found"
error_message$[101] = ""
```

```
error_message$[102] = ""
error_message$[103] = ""
error_message$[104] = ""
error_message$[105] = ""

; / Error Solutions /
error_solution$[0] = ""
error_solution$[100] = ""
error_solution$[1] = "Place a Syringe in the Space"
error_solution$[101] = "Directly Below the Robot Gripper"
error_solution$[2] = "Check to Ensure PLC is on"
error_solution$[102] = ""
error_solution$[3] = "Manually Add Solution"
error_solution$[103] = "to RGP1"
error_solution$[4] = "The Motor will Try to Home"
error_solution$[104] = "When Error is Cleared"
error_solution$[5] = "The Motor will Try to Find Syringe"
error_solution$[105] = "Top When Error is Cleared"

; / Alarm Variables /
alarm_time = 1

initialization = 0   ; set to 1 when equipment initialized
current_index = 0
simulate = 0
syrg_done = 0
running = 0
rinsing_cup = 0      ;flag for washing cup in coupling
cleaning_amount = 10  ;amount of DMF to clean the cup with
sysmsg$ = ""

; / Data Input Variables /
prot_file$ = ""
reg_file$ = ""

;Simulate Variables
stop_blinking = 0
simulate_robot = 0
log_robot = 0
log_procedure = 0
dont_wait = 0

;Calibration data
motor_safe_step = 70
syrg1_cal_asp = 14.5
syrg2_cal_asp = 40
syrg1_cal_dsp = 14.5
syrg2_cal_dsp = 40
syrg1_back = 6
syrg2_back = 5
drop_amount1 = .5
drop_amount2 = .3
dsp_drop_amount1 = .2
dsp_drop_amount2 = .1
```

```
unstick_amount1 = .5
unstick_amount2 = .1

;Create robot log file
filecloseall
filenew 1,"roblog.dat"
filecloseall

stringfree

robot_running = 0
shake_running = 0
check_motor = 0
clean_cup_done = 0

; load the current variables

call loadvars.tcl
```

```
;
;  Procedure: INSERT
;    Purpose: Insert syringes into the system
;     Author: CES
;

local choice,temp

; Determine the sequence(s) that will be inserted
numseq = 0
if (user_choice == 1)
  ; Ask the operator to input the Peptide's code and sequence
  numseq = 1
  tempcode$[1] = ""
  tempseq$[1] = ""
  inputstring tempcode$[1],"Enter the Peptide Code:"
  inputlongstring tempseq$[1],"Enter the sequence:"
  if (tempcode$[1] == "" || tempseq$[1] == "")
    numseq = 0
  endif
elseif (user_choice == 2)
  ; Get the sequence or group of sequences from a file
  call seqfile.tcl
endif

; Check whether there is enough room for insertion
if (numseq + num_syringes > 100)
  tellalarm "A maximum of " + numtostr$(100-num_syringes) + " syringes can be ad
  numseq = 0
end'

; Tell operator if insert action is cancelled
if (numseq == 0)
  tell "No syringe will be inserted"
  stop
endif

; Determine whether the Peptide codes and sequences are valid
sysmsg$ = "Checking for validity..."
screenupdate
temp = 1
while (temp <= numseq)
  if (strlen(tempcode$[temp]) > 8)
    sysmsg$ = ""
    screenupdate
    tellalarm "The code,\n" + tempcode$[temp] + ",\nis invalid."
    tell "No syringe will be inserted."
    stop
  endif
  bs_seq$ = tempseq$[temp]
  call breakseq.tcl
  if (bs_error)
    sysmsg$ = ""
    screenupdate
    tellalarm "The sequence,\n" + left$(tempseq$[temp],200) + ",\nis invalid."
```

-76-

```
      tell "... syringe will be inserted."
      stop
    endif
    if ((!strlen(tempcode$[temp])) || (!couplings))
      sysmsg$ = ""
      screenupdate
      tellalarm "Codes and sequences cannot be empty."
      tell "No syringe will be inserted."
      stop
    endif
    syr_amino_count[temp] = couplings
    tempcount[temp] = couplings
    temp = temp + 1
  endwhile

  ; Insert the sequences into the list of sequences
  sysmsg$ = "Inserting into list..."
  screenupdate
  temp = num_syringes+numseq+1
  while (temp>=current_row+numseq)
    syr_pepcode$[temp] = syr_pepcode$[temp-numseq]
    syr_size[temp] = syr_size[temp-numseq]
    syr_rack[temp] = syr_rack[temp-numseq]
    syr_loc[temp] = syr_loc[temp-numseq]
    syr_sequence$[temp] = syr_sequence$[temp-numseq]
    syr_process[temp] = syr_process[temp-numseq]
    syr_amino_step[temp] = syr_amino_step[temp-numseq]
    syr_amino_count[temp] = syr_amino_count[temp-numseq]
    syr_resin[temp] = syr_resin[temp-numseq]
    syr_subst[temp] = syr_subst[temp-numseq]
    syr_dodeprot$[temp] = syr_dodeprot$[temp-numseq]
    syr_processtime[temp] = syr_processtime[temp-numseq]
    syr_datetime$[temp] = syr_datetime$[temp-numseq]
    syr_statcolor[temp] = syr_statcolor[temp-numseq]
    temp = temp - 1
  endwhile
  temp = current_row
  while (temp <= current_row+numseq-1)
    syr_pepcode$[temp] = tempcode$[temp-current_row+1]
    syr_sequence$[temp] = tempseq$[temp-current_row+1]
    syr_amino_count[temp] = tempcount[temp-current_row+1]
    syr_size[temp] = syr_size[num_syringes+numseq+1]
    syr_rack[temp] = 0
    syr_loc[temp] = 0
    syr_process[temp] = 0
    syr_amino_step[temp] = 1
    syr_resin[temp] = syr_resin[num_syringes+numseq+1]
    syr_subst[temp] = syr_subst[num_syringes+numseq+1]
    syr_dodeprot$[temp] = syr_dodeprot$[num_syringes+numseq+1]
    syr_processtime[temp] = 0
    syr_datetime$[temp] = date$() + " " + time$()
    syr_statcolor[temp] = syr_statcolor[num_syringes+numseq+1]
    temp = temp + 1
  endwhile
  num_syringes = num_syringes + numseq
```

```
sysmsg$ =  Saving data to file..."
screenupdate
savevars $,"syringes.dat"
sy[illegible]sg$ = ""
sc[illegible]anupdate
```

```
;----------------------------------------------------------
;       Procedure: LOADVARS
;       Purpose  : Load system variables from disk and
;                  set other variables accordingly
;       Author   : MJB,CES
;----------------------------------------------------------

local a

; Define Variable Groups for data storage

;amino acid rack
clearvargroup 1
addvar 1,fast_fill,1
addvar 1,acid_code$[1],200
addvar 1,acid_conc[1],200
addvar 1,acid_avail[1],200

; Amino acids that appear in sequences
clearvargroup 2
addvar 2,num_codes
addvar 2,aalist_code$[1],200
addvar 2,aalist_req[1],200
addvar 2,aalist_avail[1],200
addvar 2,aalist_conc[1],200

; current synthetic protocol
clearvargroup 3
addvar 3,num_procs
addvar 3,prot_proc[1],200
addvar 3,prot_buf$[1],200
addvar 3,prot_vol[1],200
addvar 3,prot_air[1],200
addvar 3,prot_time[1],200
addvar 3,prot_robottime[1],200

; reagents and solutions
clearvargroup 4
addvar 4,contents$[1],10
addvar 4,sol_conc[1],10
addvar 4,sol_req[1],10
addvar 4,sol_avail[1],10

;current syringe data
clearvargroup 5
addvar 5,num_syringes
addvar 5,syr_pepcode$[1],100
addvar 5,syr_sequence$[1],100
addvar 5,syr_size[1],100
addvar 5,syr_rack[1],100
addvar 5,syr_rackloc[1],100
addvar 5,syr_loc[1],100
addvar 5,syr_grip[1],100
addvar 5,syr_amount[1],100
addvar 5,syr_resin[1],100
```

```
addvar 5,syr_subst[1],100
addvar 5,syr_dodeprot$[1],100
addvar 5,syr_process[1],100
addvar 5,syr_amino_step[1],100
addvar 5,syr_amino_count[1],100
addvar 5,syr_processtime[1],100
addvar 5,syr_datetime$[1],100

; Simulation flags
clearvargroup 6
addvar 6,simulate_robot
addvar 6,log_robot
addvar 6,log_procedure
addvar 6,dont_wait

; Current step state for restart purposes
clearvargroup 7
addvar 7,restart_info$[1],20

clearvargroup 8
addvar 8,division_factor

; Load variables from disk
erroroff
a = 0
loadvars 1,"amino.dat"
a = a + getstatus()
loadvars 2,"codes.dat"
a = a + getstatus()
loadvars 3,"lastprot.prt"
a = a + getstatus()
loadvars 4,"reagents.dat"
a = a + getstatus()
loadvars 5,"syringes.dat"
a = a + getstatus()
loadvars 6,"simulate.dat"
a = a + getstatus()
loadvars 7,"restart.dat"
a = a + getstatus()
loadvars 8,"division.dat"
a = a + getstatus()
erroron
if (a)
  ask a,"Some of the saved data did not load.\nDo you want to continue?"
  if (!a)
    terminate
  endif
endif

; Figure out the value for next_syringe
next_syringe = 1
while (next_syringe <= num_syringes && syr_process[next_syringe] > 1)
  next_syringe = next_syringe + 1
endwhile
syr_pepcode$[num_syringes+1] = "Default"
```

-80-

```
syr_sequen. .(num_syringes+1) = ""
syr_loc(num_syringes+1) = 0
syr_process(num_syringes+1) = 0

; ! 1 in the descriptions for the protocol steps
a = u
while (a <= num_procs)
  prot_desc$(a) = prot_step$(prot_proc(a))
  a = a + 1
endwhile
```

```
;--------------------------------------------------------------
;       Procedure: LOGROBOT
;       Purpose  : saves robot action and data
;       Author   : MJB
;--------------------------------------------------------------

if (!dont_wait)
  time_now = elapsed(0)
endif

fileopen 5,"roblog.dat"

stringformat logrobot$,"%s %03.01f:%02.01f:%02.01f %-8s, %s",time$(),(time_now-s
fileappend 5,logrobot$

fileclose 5
```

```
;
;        Procedure : PROTCHNG
;        Purpose   : Allow operator to add, insert, or delete step
;        Author    : MJB,CES
;

local temp

if (user_choice == 1)          ; add a procedure
  num_procs = num_procs + 1
  ; Set the selected row to be the added row
  prot_row = num_procs
  ; Set the column selection so that the user will be asked
  ; to input the type of procedure to add
  prot_col = 2
  prot_proc[prot_row] = 1
  prot_buf$[prot_row] = ""
  prot_vol[prot_row] = 0
  prot_air[prot_row] = 0
  prot_time[prot_row] = 0
  prot_robottime[prot_row] = 0
elseif (user_choice == 2)      ; insert a procedure
  screenhold
  num_procs = num_procs + 1
  temp = num_procs
  ; Move everything back in the protocol
  while (temp > prot_row)
    prot_proc[temp] = prot_proc[temp-1]
    prot_desc$[temp] = prot_desc$[temp-1]
    prot_buf$[temp] = prot_buf$[temp-1]
    prot_vol[temp] = prot_vol[temp-1]
    prot_air[temp] = prot_air[temp-1]
    prot_time[temp] = prot_time[temp-1]
    prot_robottime[temp] = prot_robottime[temp-1]
    temp = temp - 1
  endwhile
  ; Set the column selection so that the user will be asked
  ; to input the type of procedure to add
  prot_col = 2
  prot_proc[prot_row] = 1
  prot_buf$[prot_row] = ""
  prot_vol[prot_row] = 0
  prot_air[prot_row] = 0
  prot_time[prot_row] = 0
  prot_robottime[prot_row] = 0
elseif (user_choice == 3)           ; delete a procedure
  temp = prot_row
  if (temp < num_procs)
    ; Move everything forward in the protocol
    while (temp < num_procs)
      prot_proc[temp] = prot_proc[temp+1]
      prot_buf$[temp] = prot_buf$[temp+1]
      prot_vol[temp] = prot_vol[temp+1]
      prot_air[temp] = prot_air[temp+1]
      prot_time[temp] = prot_time[temp+1]
```

```
      prot_robottime(temp) = prot_robottime(temp+1)
      temp = temp + 1
    endwhile
  else
    ; The selected row cannot be the deleted row
    prot_row = prot_row - 1
  endif
  num_procs = num_procs-1
endif
```

```
;=====================================================================
;       Procedure : PROTCLIK
;       Purpose   : Handle clicking on a column
;       Author    : MJB,CES
;=====================================================================

local a,a$,choice

; Allow the user to enter the data for a cell
if (prot_col==2)        ; change the procedure
  menuarray choice,prot_step$[1],"Pick a procedure:"
  if (choice != prot_proc[prot_row] && choice>0)
    prot_proc[prot_row] = choice
    prot_buf$[prot_row] = ""
    prot_vol[prot_row] = 0
    prot_air[prot_row] = 0
    prot_time[prot_row] = 0
    prot_robottime[prot_row] = 0
    prot_changed = 1
  endif
elseif (prot_col==3)     ; enter the buffer solution name
  if (prot_choice[prot_proc[prot_row]] & param_buf)
    a$ = prot_buf$[prot_row]
    inputstring a$,"Input Buffer Name"
    if (a$ != prot_buf$[prot_row])
      prot_buf$[prot_row] = a$
      prot_changed = 1
    endif
  endif
els-'f (prot_col==4)     ; enter the buffer volume
  i  (prot_choice[prot_proc[prot_row]] & param_vol)
    a = prot_vol[prot_row]
    inputnum a,"Input Volume of Buffer or Excess"
    if (a != prot_vol[prot_row])
      prot_vol[prot_row] = a
      prot_changed = 1
    endif
  endif
elseif (prot_col==5)     ; enter the air volume
  if (prot_choice[prot_proc[prot_row]] & param_air)
    a = prot_air[prot_row]
    inputnum a,"Input Volume of Air"
    if (a != prot_air[prot_row])
      prot_air[prot_row] = a
      prot_changed = 1
    endif
  endif
elseif (prot_col==6)     ; enter the time
  if (prot_choice[prot_proc[prot_row]] & param_time)
    a = prot_time[prot_row]
    inputnum a,"Input Time (min)"
    if (a != prot_time[prot_row])
      prot_time[prot_row] = a
      prot_changed = 1
    endif
```

```
  endif
elseif (prot_col==7)      ; enter the robot use time.
  if (prot_choice[prot_proc[prot_row]] & param_robottime)
    a = prot_robottime[prot_row]
    inputint a,"Input the 'robot in use' time (sec):"
    if (a != prot_robottime(prot_row])
      prot_robottime[prot_row] = a
      prot_changed = 1
    endif
  endif
endif
```

```
;------------------------------------------------------------
;       Procedure : PROTLDSV
;       Purpose   : Load or save a protocol
;       Author    : MJB,CES
;------------------------------------------------------------

local choice,tempchanged

; Make sure there is a valid reagent specified for each step that needs one
choice = 1
while (choice <= num_procs)
  if (prot_choice[prot_proc[choice]] & param_buf)
    if (rtrim$(ltrim$(prot_buf$[choice])) == "")
      tell "You must input a buffer type for each step that needs one."
      prot_row = choice
      user_choice = 0
      stop
    endif
  endif
  choice = choice + 1
endwhile

; Make sure that the protocol starts with Ready and ends with Finished
if (prot_proc[1] != step_ready || prot_proc[num_procs] != step_finished)
  tellalarm "The protocol must begin with a 'Ready' step\nand end with a 'Finish
  user_choice = 0
  stop
endif

; i' the user is trying to quit or load, check if it needs saving
if ,_ser_choice == 5 || user_choice == 6)
  if (prot_changed)
    ask prot_changed,"Do you want to save your changes\nto a file?"
  endif
endif

if (prot_changed || user_choice == 4)
  // Save Protocol to Disk
  inputstring prot_file$,"Save Protocol As:\n(Do NOT add the Extension)"
  if (prot_file$ == "")
    user_choice = 0
    stop
  endif
  prot_file$ = prot_file$ + ".prt"
  filetest choice,prot_file$
  if (choice)
    ask choice,"That file exists.\nOvervrite?"
    if (!choice)
      user_choice = 0
      stop
    endif
  endif
  erroroff
  savevars 3,prot_file$
  i' (getstatus())
```

```
      tellalarm "Error saving file,\n"+prot file$
      user_choice = 0
      erroron
      stop
    endif
    erroron
    prot_changed = 0
  endif

  if (user_choice == 5)
    // Retrieve Protocol from Disk
    filetest choice,"*.prt"
    if (choice>0)
      filelist "*.prt",files$[1]
      menuarray choice,files$[1],"Pick a File to Load:"
      if (choice!=0)
        savevars 3,"lastprot.prt"
        erroroff
        loadvars 3,files$[choice]
        status choice
        if (choice)
          loadvars 3,"lastprot.prt"
          tellalarm "That was an invalid protocol file.\nThe last protocol was rel
          prot_changed = 1
        else
          savevars 3,"lastprot.prt"
          prot_changed = 0
        endif
        erroron
      endif
    else
      prot_changed = 1
      tell "No protocol files exist."
    endif
  endif

  if (user_choice == 6)
    user_choice = -1
  else
    user_choice = 0
  endif
```

TESTIO.TCL

```
disablekeys

; in:tialize variables
num  puts = 7
si_name$[1] = "System Start"
si_name$[2] = "Table Sensor"
si_name$[3] = "Top of Syringe"
si_name$[4] = "Low Air"
si_name$[5] = "Motor Home"
si_name$[6] = "Tumbler #1 in Position"
si_name$[7] = "Tumbler #2 in Position"
si_name$[8] = ""
si_name$[9] = ""
si_name$[10] = ""
si_name$[11] = ""
si_name$[12] = ""
si_name$[13] = ""
si_name$[14] = ""
si_name$[15] = ""
si_name$[16] = ""
si_name$[17] = ""
si_name$[18] = ""
si_name$[19] = ""
si_name$[20] = ""

; change this line to reflect the addresses of the inputs
; keep the first parameter at one
SHARK.puttable 101,0,1,2,3,4,5,6

num__tputs = 8
so_name$[1] = "Tumbler Motor #1"
so_name$[2] = "Tumbler Rack Locator #1"
so_name$[3] = "Tumbler Syringe Clamp #1"
so_name$[4] = "Tumbler Motor #2"
so_name$[5] = "Tumbler Rack Locator #2"
so_name$[6] = "Tumbler Syringe Clamp #2"
so_name$[7] = "2.5ml Syringe Locator"
so_name$[8] = "Buzzer"
so_name$[9] = ""
so_name$[10] = ""
so_name$[11] = ""
so_name$[12] = ""
so_name$[13] = ""
so_name$[14] = ""
so_name$[15] = ""
so_name$[16] = ""
so_name$[17] = ""
so_name$[18] = ""
so_name$[19] = ""
so_name$[20] = ""
so_address[1] = 220
so_address[2] = 223
so_address[3] = 222
so_ -dress[4] = 221
```

-87-

```
so_address[5] = 241
so_address[6] = 240
so_address[7] = 227
so_address[8] = 226
so_address[9] = -1
so_address[10] = -1
so_address[11] = -1
so_address[12] = -1
so_address[13] = -1
so_address[14] = -1
so_address[15] = -1
so_address[16] = -1
so_address[17] = -1
so_address[18] = -1
so_address[19] = -1
so_address[20] = -1

a = 1
while (a <= numoutputs)
  readbit so_address[a],so_on[a]
  a = a + 1
endwhile

stopme = 0
screenison = 0
doterminal = 0
terminal_running = 0
choice = 0
while (!stopme)

  if (choice)
    writebit so_address[choice],!so_on[choice]
    so_on[choice] = !so_on[choice]
    choice = 0
  endif

  a = 1
  while (a <= numinputs-10)
    readbit a+100,si_on[a],si_on[a+1],si_on[a+2],si_on[a+3],si_on[a+4],si_on[a+5
    a = a + 10
  endwhile
  while (a <= numinputs-5)
    readbit a+100,si_on[a],si_on[a+1],si_on[a+2],si_on[a+3],si_on[a+4]
    a = a + 5
  endwhile
  while (a <= numinputs)
    readbit a+100,si_on[a]
    a = a + 1
  endwhile

  if (doterminal)
    if (!terminal_running)
      run terminal.tcl
    endif
    doterminal = 0
```

```
    endif

    if (!screenison)
      screenon "testio"
      enablekeys
      screenison = 1
    endif

endwhile

request "robot"
ROBOT.command "NOMANUAL"
free

screenback
```

```
;=============================================================
;       Procedure : PROTOCOL
;       Purpose   : Allow operator to change protocol data
;       Author    : MJB,CES
;=============================================================

local choice,simulate_change,temp

simulate_change = simulate_robot+(dont_wait*2)+(log_robot*4)+(log_procedure*8)
prot_col = 0
prot_row = 0
prot_changed = 0
prot_file$ = ""
prot_proc[0] = 0
prot_proc[100] = 100
screenon "protocol"

user_choice = 0
while (user_choice != -1)

  // Clicked on Synthetic Protocol Box
  if (prot_col)
    screenupdate
    if (next_syringe <= 1)
      call protclik.tcl
    endif
    updatetable
    screenupdate
    prot_col = 0
  e' sif (user_choice == 7)
    . Change division factor
    temp = division_factor
    inputnum temp,"Enter the new division factor:"
    if (temp > 0 && temp <= 99.99)
      division_factor = temp
      savevars 8,"division.dat"
      screenupdate
    endif
    user_choice = 0
  elseif ((user_choice && next_syringe <= 1) || user_choice == 6)
    if (user_choice >= 1 && user_choice <= 3)
      call protchng.tcl
      updatetable
      screenupdate
      user_choice = 0
    elseif (user_choice >= 4 && user_choice <= 6)
      call protldsv.tcl
      updatetable
      screenupdate
    endif
  endif

endwhile

;Save protocol to permanent file
```

```
savevars 3,"lastprot.prt"

user_choice = 0
if (simulate_change!=simulate_robot+(dont_wait*2)+(log_robot*4)+(log_procedure*8
  savevars 6,"simulate.dat"
  initialization = 0
endif

; Fill in the descriptions for the protocol steps
a = 0
while (a <= num_procs)
  prot_desc$[a] = prot_step$[prot_proc[a]]
  a = a + 1
endwhile

screenback
```

```
;
; Procedure: PUTHOLD
;   Purpose: put a syringe in the holding rack
;    Author: MJB
;     Notes:
;

;
;        set syringe = 1 for 10ml , 2 for 2.5ml
;            grip    = 1 for move grip , 2 for stroke grip
;             nest    = 1 for nest 1, 2 for nest 2
;

local a

if (syringe==1)
  rack_location = 1
  rack_type = 8
else
  rack_location = 1
  rack_type = 9
endif

if (log_robot)
  logrobot$ = "puthold -- syringe : " + numtostr$(syringe) + " grip : " + numtos
  call logrobot.tcl
endif

if (simulate_robot)
  i' '!getting_rt)
    call datrack.tcl
  endif
  stop
endif

request "ROBOT"

ROBOT.speed sp_fast

;a$ = "mixsafe"+ numtostr$(grip)

;ROBOT.move a$

ROBOT.speed sp_mod

a$ = "tmp" + numtostr$(syringe) + "_" + numtostr$(grip) + "_" + numtostr$(nest)

ROBOT.move a$

if (grip==1)
  ROBOT.speed sp_slow
else
  ROBOT.speed sp_vslow
endif
```

```
a$ = "tlw" + numtostr$(syringe) + "_" + numtostr$(g..p) + "_" + numtostr$(nest)
ROBOT.move_s a$
ROBOT.finish

ROBOT.open
if (grip==1&&syringe==2)
  delay 500
else
  delay 250
endif

if (grip==1)
  ROBOT.speed sp_med
else
  ROBOT.speed sp_slow
endif

if (grip==2)
  run homemtr.tcl
endif

;check motor pulses
if (grip==2&&check_motor)
  while (!home_done)
  endwhile
  call findtop.tcl
  run homemtr.tcl
endif

a$ = "tmp" + numtostr$(syringe) + "_" + numtostr$(grip) + "_" + numtostr$(nes
ROBOT.move_s a$

ROBOT.finish

if (syringe==2)
  SHARK.writebit o_syringe_locate,1
  delay 1500
  SHARK.writebit o_syringe_locate,0
endif

delay 500

if (grip==1&&syringe==2)

  ROBOT.speed sp_mslow

  a$ = "tlw" + numtostr$(syringe) + "_" + numtostr$(grip) + "_" + numtostr$(nes
  ROBOT.move_s a$

  ROBOT.finish

  ROBOT.jog_s 0,0,.25
  ROBOT.finish

  ROBOT.close
```

```
   delay 250

   ROBOT.open
   delay 250

   a$ = "tmp" + numtostr$(syringe) + "_" + numtostr$(grip) + "_" + numtostr$(nest
   ROBOT.move_s a$

   ROBOT.finish

endif

a$ = "mixsafe" + numtostr$(grip)

;ROBOT.speed sp_fast

;ROBOT.move a$

if (grip==2)
  while (!home_done)
  endwhile
  run safemtr.tcl
endif

if (!getting_rt)
  call datrack.tcl
endif

free
```

```
;
;   PUTRACK
;   put a bottle in a rack from the any safe position
;

/**************************
USE OF PROCEDURE:

before calling this procedure set the variable:
    rack_type  to reflect the type of bottle that is being picked up
    robot_location = the location in the rack

the following variables should be set in the gen_init.tcl procedure:

    row_size[rack_type] = number of locations that are parallel to the track
    rack_jog[rack_type] = distance to jog down to get bottle
    track_jog[rack_type] = distance between the locations
    rack_safe$[rack_type] = safe position
    rack_im_1$[rack_type] = intermediate point #1
    rack_im_2$[rack_type]
    rack_im_3$[rack_type]
    rack_high$[rack_type] = high point names for this rack
            (the procedure adds the location number to this name so that
            the robot moves to the first location in a row and moves the
            track to the appropriate final location. So rack_high$ should
            look like: "SAM_HI_")
NOTES:
  If there are more racks than bottle types, you can have many of these
  types of procedures for each type. For example, create a new procedure
  called getrack1.tcl and call each of the arrays by a different name. So
  rack_high$[rack_type] could be renamed rack1_high$[rack_type], etc.

**************************************/

if (log_robot)
  logrobot$ = "putrack -- rack: " + numtostr$(rack_type) + " loc: " + numtostr$
  call logrobot.tcl
endif

if (simulate_robot)
  if (!getting_rt)
    call datrack.tcl
  endif
  stop
endif

request "ROBOT"

ROBOT.speed sp_vfast

if(rack_type==5)
  tum1_continue=0
  while (!tum1_latched)
  endwhile
```

```
elseif (rack_type==6)
  tum2_continue=0
  while (!tum2_latched)
  endwhile
endif

found = 0
robot_i = 1
while (!found)
  if (rack_location <= robot_i * row_size(rack_type))

    if ((rack_type==3)||(rack_type==4))
      ROBOT.move "rack_in"
    endif

    rack_temp$ = rack_high$(rack_type) + numtostr$(robot_i)
    ROBOT.move rack_temp$
    ROBOT.finish
    found = 1
  else
    robot_i = robot_i + 1
  endif
endwhile

ROBOT.speed sp_fast

jog_temp = track_jog(rack_type)*((row_size(rack_type)*robot_i) - rack_location)
ROBOT.joint 6,-jog_temp
ROBOT.finish

if ((rack_type==5)||(rack_type==6)||(rack_type==3))
  ROBOT.speed sp_vslow
else
  ROBOT.speed sp_slow
endif

if (rack_type==7)
  ROBOT.jog_s 0,0,(-rack_jog(rack_type)+.5)
else
  ROBOT.jog_s 0,0,-rack_jog(rack_type)
endif
ROBOT.finish
ROBOT.open
delay 500
ROBOT.speed sp_slow
ROBOT.jog_s 0,0,rack_jog(rack_type)
ROBOT.finish
ROBOT.speed sp_fast

if (!getting_rt)
  call datrack.tcl
endif

if (rack_type==5)
  tum1_continue=1
```

```
elseif (ra.._type==6)
  turn2_continue=1
endif

fi
```

```
;
; Procedure: PUT_TUM
;   Purpose: Put syringe in tumbler
;    Author: MJB
;

if (syr_rack[current_index]!=10)
  if (syr_size[current_index]==10)
    syringe = 1
  else
    syringe = 2
  endif
  grip = 1
  nest = 1
  call gethold.tcl
  syr_rack[current_index] = 10
  syr_rackloc[current_index] = 0
  syr_grip[current_index] = 1
endif

;change grip if needed
if (syr_grip[current_index]==2)
  if (syr_size[current_index]==10)
    syringe = 1
  else
    syringe = 2
  endif
  grip = 2
  nest = 1
  c 'l puthold.tcl
  c..l stopchek.tcl
  grip = 1
  nest = 1
  call gethold.tcl
  call stopchek.tcl
endif

;find next available tumbling rack space
if (syr_size[current_index]==10)
  rack_kind = 5
else
  rack_kind = 6
endif
tum_current = 0
old_tum = 1
syr_rackloc[current_index] = 0

while (syr_rackloc[current_index]==0)

  no_index = 0
  tum_current = 0

  while (!no_index)
    narrayfind tum_current,syr_rackloc[tum_current+1],old_tum
    if (tum_current!=-1)
```

-100 -

```
      if (syr_rack(tum_current)==rack_kind)
        no_index = 1
      endif
    elseif (tum_current==-1)
      syr_rackloc[current_index] = old_tum
      no_index = 1
    endif
  endwhile

  old_tum = old_tum + 1

endwhile

if (syr_size[current_index]==10)
  syr_rack[current_index] = 5
else
  syr_rack[current_index] = 6
endif

;place syringe in tumbling rack
rack_type = syr_rack[current_index]
rack_location = syr_rackloc[current_index]
call putrack.tcl
call stopchek.tcl
```

```
;----------------------------------------------------------
;  Procedure: MONITOR
;    Purpose: Main procedure for data input
;     Author: CES
;----------------------------------------------------------

user_choice = 0
current_row = num_syringes+1
last_row = current_row
screenon "monitor"
enablekeys

running = 1
run robot.tcl
run shake.tcl

while (running)
  if (user_choice != 0)
    if (user_choice == 1 || user_choice == 2)
      ; Insert syringe(s)
      if (num_syringes >= 100)
        tellalarm "No more syringes may be added."
      else
        disablekeys
        screenhold
        call insert.tcl
        updatetable
        screenupdate
        enablekeys
      endif
      user_choice = 0
    elseif (user_choice == 3)
      disablekeys
      delete_row = current_row
      call delete.tcl
      updatetable
      screenupdate
      enablekeys
      user_choice = 0
    elseif (user_choice == 4)
      stringrequest "CALCULATING"
      call calc.tcl
      stringfree "CALCULATING"
      user_choice = 0
    elseif (user_choice == 5)
      call reagents.tcl
      user_choice = 0
    elseif (user_choice == 6)
      call amino.tcl
      user_choice = 0
    elseif (user_choice == 7)
      ; clear all syringes...only works if 'initialization = 0 '
      disablekeys
      call clearall.tcl
      enablekeys
```

```
      user_choice = 0
    elseif (user_choice == 8)
      disablekeys
      call snapshot.tcl
      enablekeys
      user_choice = 0
    elseif (user_choice == 20)
      call equpinit.tcl
      screenupdate
      enablekeys
      user_choice = 0
    elseif (user_choice == 21)
      ; change protocol
      call protocol.tcl
      user_choice = 0
    elseif (user_choice == 22)
      call testio.tcl
      user_choice = 0
    elseif (user_choice == -1)
      ; Verify termination of system
      ask user_choice,"Are you sure you want to terminate?\n"
      if (user_choice)
        call stopmtr.tcl
        terminate
      endif
      user_choice = 0
    endif
  elseif (current_col || current_row != last_row)
    if (current_row == last_row)
      if (current_row <= num_syringes+1 && syr_process(current_row]<=1)
        disablekeys
        screenupdate
        screenhold
        call syrinfo.tcl
        updatetable
        screenupdate
        enablekeys
      endif
    else
      last_row = current_row
    endif
    current_col = 0
  endif
endwhile
screenon "logo"
initialization = 0
```

```
;---------------------------------------------------------------
;        Procedure  : NEXTSTEP
;        Purpose    : Figure out the next step for a syringe
;        Author     : MJB,CES
;        Notes      : Input --  ns_process
;                               ns_amino
;                               ns_dodep$
;                      Output -- ns_process
;                               ns_amino
;---------------------------------------------------------------

; If it is the end of an amino acid, figure out what the next process
; should be
if (prot_proc[ns_process] == step_nextamino)
  ns_amino = ns_amino + 1
  if (ns_amino <= couplings)
    ns_process = 2
  else
    ns_process = ns_process + 1
  endif

; If it is the Stop if no deprotection, figure out if we are done
elseif (prot_proc[ns_process] == step_stopifnofinaldep)
  if (ns_dodep$ == "Y")
    ns_process = ns_process + 1
  else
    ns_process = num_procs
  endif

; c'herwise, just add one to the process number
eli
  ns_process = ns_process + 1
endif

setstringarray restart_info$[1],"",-1
savevars 7,"restart.dat"
```

```
;=================================================================
;       Procedure : REAGENTS
;       Purpose   : Allow operator to change reagents
;       Author    : MJB
;=================================================================

local a$,a,change

disablekeys
; Set the colors of the volumes -- set the ones that have their required
; greater than their available to red.
a = 1
while (a <= 4)
  if (sol_req[a] > sol_avail[a])
    sol_color[a] = 4
  else
    sol_color[a] = 15
  endif
  a = a + 1
endwhile
sol_col = 0
sol_row = 0
screenon "reagents"
enablekeys
user_choice = 0
while (user_choice != -1)

  if (sol_row)
    if (sol_col==2)
      a$ = contents$(sol_row)
      inputstring a$,"Input Probe Contents"
      if (a$ != contents$[sol_row] && a$ != "")
        contents$(sol_row) = a$
        change = 1
      endif
    elseif (sol_col==3)
      a = sol_conc[sol_row]
      inputnum a,"Input Concentration"
      if (a != sol_conc[sol_row] && a > 0)
        sol_conc[sol_row] = a
        change = 1
      endif
    elseif (sol_col==5)
      a = sol_avail[sol_row]
      inputnum a,"Input Available Volume"
      if (a != sol_avail[sol_row] && a >= 0)
        sol_avail[sol_row] = a
        if (sol_avail[sol_row] >= sol_req(sol_row))
          sol_color[sol_row] = 15
        else
          sol_color[sol_row] = 4
        endif
        change = 1
      endif
  endif
```

```
      updat..able
      sol_row = 0
      sol_col = 0
    elseif (user_choice == 1)
      savevars 4,"reagents.dat"
      user_choice = -1
    elseif (user_choice == 2)
      user_choice = -1
      if (change)
        ask a,"Exit without saving?"
        if (a)
          loadvars 4,"reagents.dat"
        else
          user_choice = 0
        endif
      endif
    endif

  endwhile

  screenback
```

```
;-----------------------------------------------------------------
;  Procedure: ROBHOME
;    Purpose: Home the robot.
;     Author: MJB,CES
;      Notes:
;-----------------------------------------------------------------

local choice

request "ROBOT"
sysinit_message$ = ""
robot_ready = 1
screenupdate

; Arm Power
tell "Make sure Robot Arm Power is on."
robot_ready = robot_ready | 2
ROBOT.cleararm
screenupdate

; Home the robot
ask choice,"Is the Robot homed?"
if (choice)
  robot_ready = robot_ready | 4
  screenupdate
  ask choice, "Is the Robot in a\nsafe position?"
  if (!choice)
    ROBOT.manualc
    sysinit_message$ = "Robot is in manual mode"
    screenupdate
    tell "The Robot is now in manual mode.\nUse teach pendant to place it\nin a
    ask choice, "Is the Robot in a safe position?\n(<No> cancels initialization)
    if (!choice)
      robot_ready = 0
      stop
    endif
  endif
else
  /* Ask if the robot is in its HOMEHERE position */
  ask choice, "Is the Robot in its\nHOMEHERE position?"
  if (choice)
    ask choice, "Are you sure?"
  endif
  if (!choice)
   /* get the robot in its home bounds */
   ROBOT.manual
   sysinit_message$ = "Robot is in manual mode"
   screenupdate
   tell "The robot is now in manual mode.\nUse the teach pendant to place\nit in
  endif
  /* home robot */
  choice = 0
  ask choice,"Ready to home?"
  if (!choice)
    robot_ready = 0
```

```
      stop
    endif
    sysinit_message$ = "Homing sequence in progress"
    disablekeys
    ROBOT.command "NOMANUAL"
    ROBOT.command "NOLIMP"
    ROBOT.speed vsafe_speed
    ROBOT.home
    robot_ready = robot_ready | 4
  endif

  ; Move to VERIFY position
  sysinit_message$ = "Moving to VERIFY..."
  screenupdate
  ROBOT.command "NOMANUAL"
  ROBOT.speed sp_med
  ROBOT.move "VERIFY"
  ROBOT.finish
  sysinit_message$ = ""
  ask choice,"Is the robot at its\nVERIFY position?\n(<No> will cancel initializa
  if (!choice)
    robot_ready = 0
    stop
  endif
  ROBOT.open
  robot_ready = robot_ready | 8
  free

/* ROBOT is at VERIFY Position */
```

```
;=====================================================================
;  Procedure: ROBOT
;    Purpose: Control the robot movements
;     Author: MJB,CES
;      Notes:
;=====================================================================

if (robot_running)
  stop
endif

local last_current,a

robot_running = 1

system_paused = 0
error = 0
pause_system = 1
current_index = 0
timer start_time
time_now = start_time

while (running)

  if (initialization)

    stringrequest "USING SYRINGES"

    ; If no syringes are in the system, set the start time to zero
    if (num_syringes == 0)
      timer start_time
      time_now = start_time
    endif

    ; check if robot has been paused
    call stopchek.tcl

    ; check if all the syringes must be cleared
    if (clear_all_syringes)
      call clearall.tcl
    endif

    ;find next syringe that robot will work on:  current_index
    call whattodo.tcl

    ; while the robot has a syringe, perform the next step for the
syringe
    while (current_index > 0)
      syr_statcolor[current_index] = 1
      last_current = current_index
      updatetable
      call dostep.tcl
      syr_statcolor[last_current] = 0
      updatetable
```

```
      if (current_index)
        stringrequest "NEXT STEP"
        ns_process = syr_process[current_index]
        ns_amino = syr_amino_step[current_index]
        ns_dodep$ = syr_dodeprot$[current_index]
        ; Find the codes for the syringe
        bs_seq$ = syr_sequence$[current_index]
        call breakseq.tcl
        call nextstep.tcl
        syr_process[current_index] = ns_process
        syr_amino_step[current_index] = ns_amino
        stringfree "NEXT STEP"
      endif
      savevars 5,"syringes.dat"
    endwhile

    stringfree "USING SYRINGES"

  endif

endwhile
robot_running = 0
```

```
;-------------------------------------------------------------
;       Procedure: ROTATE
;       Purpose  : rotate syringe
;       Author   : MAB
;-------------------------------------------------------------

;=============================================================
;           set   rot_time = time to rotate in seconds
;=============================================================

if (log_robot)
  logrobot$ = "rotate -- time: " + numtostr$(rot_time)
  call logrobot.tcl
endif

if (simulate_robot)
  stop
endif

local wait

request "ROBOT"

ROBOT:speed sp_slow

timer wait

ROBOT.speed 150
while (elapsed(wait)<=rot_time)
  P--OT.command "VIA MIXING,MIXSAFE2,MIXING"
  k .OT.command "VIA MIXSAFE2,MIXING,MIXSAFE2"
  ROBOT.finish
;  ROBOT.joint 4,90
;  ROBOT.joint 4,-90
endwhile
ROBOT.speed sp_slow

free
```

```
;--------------------------------------------------------------
;       Procedure: SAFEMTR
;       Purpose  : move motor to a safe position
;       Author   : MJB
;--------------------------------------------------------------
direction = 1

motor_steps = motor_safe_step

call strkmtr.tcl
```

```
;=================================================================
;   NAME: SHAKE
;   PURPOSE:  To Shake the appropriate tumblers
;   AUTHOR: SBP
;=================================================================

if (shake_running)
  stop
endif

local a

shake_running = 1

tum1_latched = 1
tum2_latched = 1
tum1_continue = 1
tum2_continue = 1
while ((!initialization) && (running))
endwhile

while (running)

  if (!simulate_robot)

    if (tum1_latched && tum1_continue)
      ; Check if there is anything in the rack
      narrayfind a,syr_rack[1],5
      if (a != -1)
        ; turn tumbler 1 on
        SHARK.writebit o_tum1_clamp,1
        delay 250
        SHARK.writebit o_tum1_latch,0
        delay 500
        SHARK.writebit o_tum1_start,1
      else
        tum1_continue = 0
      endif
      tum1_latched = 0
    endif

    if (tum1_continue == 0 && tum1_latched == 0)
      ; turn tumbler 1 off
      SHARK.writebit o_tum1_latch,0
      delay 250
      SHARK.writebit o_tum1_start,1
      SHARK.readbit i_sense_tum1,a
      while (!a)
        SHARK.readbit i_sense_tum1,a
      endwhile
      SHARK.writebit o_tum1_start,0
      delay 250
      SHARK.writebit o_tum1_latch,1
      delay 250
      SHARK.writebit o_tum1_clamp,0
```

```
      tum1_latched = 1
    endif

    if (tum2_latched && tum2_continue)
      ; Check if there is anything in the rack
      narrayfind a,syr_rack[1],6
      if (a != -1)
        ; turn tumbler 2 on
        SHARK.writebit o_tum2_clamp,1
        delay 250
        SHARK.writebit o_tum2_latch,0
        delay 500
        SHARK.writebit o_tum2_start,1
      else
        tum2_continue = 0
      endif
      tum2_latched = 0
    endif

    if (tum2_continue== 0 && tum2_latched == 0)
      ; turn tumbler 2 off
      SHARK.writebit o_tum2_latch,0
      delay 250
      SHARK.writebit o_tum2_start,1
      SHARK.readbit i_sense_tum2,a
      while (!a)
        SHARK.readbit i_sense_tum2,a
      endwhile
      SHARK.writebit o_tum2_start,0
      delay 250
      SHARK.writebit o_tum2_latch,1
      delay 250
      SHARK.writebit o_tum2_clamp,0
      tum2_latched = 1
    endif

  endif

endwhile
shake_running = 0
```

```
;---------------------------------------------------------
;  Procedure: SNAPSHOT
;    Purpose: Take a snapshot of the syringe data
;     Author: CES
;      Notes:
;---------------------------------------------------------

local file$

file$ = "snapshot.dat"
inputstring file$,"Enter the file name:"
if (file$ == "")
  stop
endif

erroroff
filenew 6,file$
if (getstatus())
  tellalarm "Invalid file name!"
  stop
endif
erroron

sysmsg$ = "Waiting for robot..."
screenupdate
stringrequest "SYRFILE"
sysmsg$ = "Writing to file..."
screenupdate

sf  ow = 1
wh. : (sf_row <= num_syringes)
  call syrfile.tcl
  sf_row = sf_row + 1
endwhile

fileclose 6
sysmsg$ = ""
screenupdate

stringfree "SYRFILE"
```

```
;=====================================================================
;       Procedure: STARTUP
;       Purpose  : startup system
;       Author   : MJB
;=====================================================================

enablekeys
call init.tcl
if (next_syringe <= 1)
  screenon "logo"
else
  call restart.tcl
  run monitor.tcl
endif
```

```
;------------------------------------------------------------------
; Procedure:   STOPCHEK
; Purpose:     checks to see if pause system has been hit
; Author:      MJB
;------------------------------------------------------------------

; Free this string because it is ok to update the syringe list
stringfree "USING SYRINGES"
if (pause_system)
  if (!simulate_robot)
    request "ROBOT"
    ROBOT.manualc
    free
  endif
  system_paused = 1
  screenupdate
  while (pause_system)
  endwhile
  if (!simulate_robot)
    request "ROBOT"
    ROBOT.command "NOMANUAL"
    free
  endif
  system_paused = 0
  screenupdate
endif
; request syringe list again
stringrequest "USING SYRINGES"
```

```
;=====================================================================
;       Procedure: SYR_ASP
;       Purpose  : move syringe to cup and aspirate solution
;       Author   : MJB
;=====================================================================

;=====================================================================
;         set    cup = 1 for RGP1, 2 for RGP2
;                syringe = 1 for 10ml , 2 for 2.5ml
;                asp_amount = amount of solution to aspirate
;                air_amount = amount of air to draw in
;=====================================================================

if (log_robot)
  logrobot$ = "syringe aspirate -- cup: " + numtostr$(cup) + " syringe: " + num
  call logrobot.tcl
  logrobot$ = "                    -- amount: " + numtostr$(asp_amount) + " air am
  call logrobot.tcl
endif

if (simulate_robot)
  stop
endif

request "ROBOT"

ROBOT.speed sp_fast

ROBOT.move "mixsafe2"

a$   "mixs" + numtostr$(cup) + numtostr$(syringe)

ROBOT.move a$

ROBOT.speed sp_vslow

while (!disp_done)
endwhile

ROBOT.jog_s 0,0,-4

ROBOT.finish

delay 500

/* Program to pull up syringe for solution amount */
if (syringe==1)
  motor_steps = (asp_amount+air_amount)*syrg1_cal_asp
  syr_amount[current_index] = syr_amount[current_index] + asp_amount + air_amou
else
  motor_steps = (asp_amount+air_amount)*syrg2_cal_asp
  syr_amount[current_index] = syr_amount[current_index] + asp_amount + air_amou
endif
if ((motor_steps-int(motor_steps))>=.5)
  motor_steps = int(motor_steps) + 1
```

```
else
  motor_steps = int(motor_steps)
endif
direction = 0
call strkmtr.tcl

/*================================*/

;ROBOT.jog_s 0,0,4
;ROBOT.finish

delay 500

savevars 5,"syringes.dat"

ROBOT.speed sp_mslow

ROBOT.jog_s 0,0,4
ROBOT.finish

ROBOT.speed sp_fast

a$ = "mixs" + numtostr$(cup) + numtostr$(syringe)

ROBOT.move a$

ROBOT.move "mixsafe2"

free
```

```
;--------------------------------------------------------------
;       Procedure: SYR_DSP
;       Purpose  : move syringe to cup and dispense solution
;       Author   : MJB
;--------------------------------------------------------------

;--------------------------------------------------------------
;         set     cup = 1 for RGP1, 2 for RGP2
;                 dsp_amount = amount of solution to dispense
;--------------------------------------------------------------

if (log_robot)
  logrobot$ = "syringe dispense -- cup: " + numtostr$(cup) + " amount: " + numto
  call logrobot.tcl
endif

if (simulate_robot)
  stop
endif

request "ROBOT"

ROBOT.speed sp_fast

ROBOT.move "mixsafe2"

ROBOT.speed sp_med

a$ = 'mixs" + numtostr$(cup) + numtostr$(syringe)

ROBOT.move a$

ROBOT.speed sp_vslow

ROBOT.jog_s 0,0,-3.25

ROBOT.finish

delay 500

/* Program to dispense from a syringe*/
if (syringe==1)
  motor_steps = dsp_amount*syrg1_cal_dsp
else
  motor_steps = dsp_amount*syrg2_cal_dsp
endif
if ((motor_steps-int(motor_steps))>=.5)
  motor_steps = int(motor_steps) + 1
else
  motor_steps = int(motor_steps)
endif
direction = 1
call strkmtr.tcl
```

```
/*-----------------------------------------------*/

;/* Program to suck up drop from tip*/
if (syringe==1)
  motor_steps = dsp_drop_amount1*syrg1_cal_dsp
else
  motor_steps = dsp_drop_amount2*syrg2_cal_dsp
endif
if ((motor_steps-int(motor_steps))>=.5)
  motor_steps = int(motor_steps) + 1
else
  motor_steps = int(motor_steps)
endif
direction = 0
call strkmtr.tcl

/*-----------------------------------------------*/

ROBOT.speed sp_mslow

ROBOT.jog_s 0,0,3.25
ROBOT.finish

ROBOT.speed sp_med

ROBOT.move "mixsafe2"

free
```

FOCUS PAGE

Title:
Apparatus and Method for Multiple Synthesis of Organic Compounds on Polymer Support

Authors:
Krchnak et al

Attorney Docket No.:
7156-068

```
;-----------------------------------------------------------
;  Procedure: SYRFILE
;    Purpose: Write a syringe's data to a file
;     Author: CES
;      Notes: The file number will be 3 upon entry
;             sf_row will be the syringe to log
;-----------------------------------------------------------

local a$

stringformat a$,"Pepcode: %-15.15s Size: %s ml",syr_pepcode$[sf_row],numtostr$(s
fileappend 6,a$
a$ = "Sequence: " + syr_sequence$[sf_row]
fileappend 6,a$
if (prot_proc[syr_process[sf_row]] != step_finished)
  stringformat a$,"Current Amino: %2.0lf    Resin: %4.1lf    Subst: %4.2lf    Fi
  fileappend 6,a$
  if (syr_loc[sf_row])
    stringformat a$,"Status:  %-13.13s Location:  In Input Rack (Loc %.0lf)",pro
  elseif (syr_rack[sf_row] == 5 || syr_rack[sf_row] == 6)
    stringformat a$,"Status:  %-13.13s",prot_step$[prot_proc[syr_process[sf_row]
  else
    stringformat a$,"Status:  %-13.13s",prot_step$[prot_proc[syr_process[sf_row]
  endif
  fileappend 6,a$
  a$ = "Start: " + syr_datetime$[sf_row]
  fileappend 6,a$
else
  stringformat a$,"Resin: %4.1lf    Subst: %4.2lf    Final Deprot: %s",syr_resin
  fileappend 6,a$
  a$ = "Start: " + syr_datetime$[sf_row] + "   Finished: " + date$() + " " + tim
  fileappend 6,a$
endif
fileappend 6,""
```

```
;
; Procedure: SYRINFO
;   Purpose: User wants to change syringe info
;    Author: CES
;
local a,a$,choice,temp,oldchoice

; change the peptide code
if (current_col == 1 && current_row <= num_syringes)
  a$ = syr_pepcode$[current_row]
  inputstring a$,"Enter the peptide's code:"
  if (a$ == "" || strlen(a$) > 8)
    tellalarm "That code is invalid."
  elseif (a$ != syr_pepcode$[current_row])
    syr_pepcode$[current_row] = a$
    savevars 5,"syringes.dat"
  endif

; change the peptide sequence
elseif (current_col == 2 && current_row <= num_syringes)
  a$ = syr_sequence$[current_row]
  inputlongstring a$,"Enter the peptide's sequence:"
  bs_seq$ = a$
  call breakseq.tcl
  if (bs_error || (!couplings))
    tellalarm "That sequence is invalid."
  elseif (a$ != syr_sequence$[current_row])
    syr_amino_count[current_row] = couplings
    syr_sequence$[current_row] = a$
    yr_process[current_row] = 0
    alc_needed = 1
    savevars 5,"syringes.dat"
  endif

; change the syringe size
elseif (current_col == 4)
  menu choice,"Pick a size:","10 ml","2.5 ml"
  if (choice && syr_size[current_row] != 10-(7.5*(choice-1)))
    syr_size[current_row] = 10-(7.5*(choice-1))
    syr_loc(current_row] = 0
    syr_rack[current_row] = 0
    syr_rackloc[current_row] = 0
    syr_process[current_row] = 0
    savevars 5,"syringes.dat"
  endif

; change the syringe location
elseif (current_col == 5 && current_row <= num_syringes)
  choice = syr_loc[current_row]
  inputint choice,"Enter the location:"
  if (choice != syr_loc[current_row])
    if (choice)
      if (choice < 1 || choice > 30)
        tellalarm "The location must be between 1 and 30"
        sysmsg$ = ""
```

```
          screenupdate
          stop
        endif
        sysmsg$ = "Checking rack..."
        screenupdate
        temp = 1
        while (temp <= num_syringes)
          if (syr_loc[temp] == choice && temp != current_row && syr_size[temp] ==
            tellalarm "That location is already used!"
            sysmsg$ = ""
            screenupdate
            stop
          endif
          temp = temp + 1
        endwhile
        sysmsg$ = ""
        screenupdate
      endif
      if (syr_loc[current_row] == 0 && choice > 0)
        calc_needed = 1
      elseif (choice == 0)
        syr_process[current_row] = 0
      endif
      syr_loc[current_row] = choice
      savevars 5,"syringes.dat"
    endif

  ; Change resin amount
  elseif (current_col == 6)
    a = syr_resin[current_row]
    inputnum a,"Enter the resin amount:"
    if (a != syr_resin[current_row])
      syr_resin[current_row] = a
      if (syr_process[current_row] > 0)
        syr_process[current_row] = 0
        calc_needed = 1
      endif
      savevars 5,"syringes.dat"
    endif

  ; Change substituion amount
  elseif (current_col == 7)
    a = syr_subst[current_row]
    inputnum a,"Enter the subst. amount:"
    if (a != syr_subst[current_row])
      syr_subst[current_row] = a
      if (syr_process[current_row] > 0)
        syr_process[current_row] = 0
        calc_needed = 1
      endif
      savevars 5,"syringes.dat"
    endif

  ; Do final deprotection?
  elseif (current_col == 8)
```

```
 if (syr_dodeprot$[current_row] == "Y")
   syr_dodeprot$[current_row] = "N"
 else
   syr_dodeprot$[current_row] = "Y"
   if
 if (syr_process[current_row] > 0)
   syr_process[current_row] = 0
   calc_needed = 1
 endif
 savevars 5,"syringes.dat"
endif
```

```
;-----------------------------------------------------------------
;  Procedure: SYRINIT
;    Purpose: Initialize the syringe pump
;     Author: MJB,CES
;      Notes:
;-----------------------------------------------------------------

local choice,temp

syringe_ready = 1
screenupdate

; Check syringe pump power
sysinit_message$ = "Checking Syringe Pump Power"
screenupdate
call syrpower.tcl
if (!sp_power)
  syringe_ready = 0
  erroron
  stop
endif
syringe_ready = syringe_ready | 2

; Fill the reservior
tell "Fill the Syringe Pump reservoir."
syringe_ready = syringe_ready | 4
screenupdate

; Prime the syringes
ask  hoice,"Do you want to prime the syringe pumps?"
if ( noice)
  tell "Prime them now."
  tell "Turn the pumps off and then on."
endif
sysinit_message$ = "Checking Syringe Pump Power"
screenupdate
call syrpower.tcl
if (!sp_power)
  syringe_ready = 0
  erroron
  stop
endif
syringe_ready = syringe_ready | 8
screenupdate
erroron
```

```
;------------------------------------------------------------
;        Procedure: WASHING
;        Purpose  : executes robot actions for a washing
;        Author   : MJB
;------------------------------------------------------------

local a,buf_place,infoindex

infoindex = 1

;find buffer in reagent table
sarrayfind buf_place,contents$[1],prot_buf$[syr_process[current_index]]

;if syringe just started, get from input rack
if (syr_rack[current_index]!=8&&syr_rack[current_index]!=10&&syr_rack[current_in
  if (syr_size[current_index]==10)
    rack_type = 1
  else
    rack_type = 2
  endif
  rack_location = syr_rackloc[current_index]
  call getrack.tcl
  syr_grip[current_index] = 1
  call stopchek.tcl
endif

;switch grip on syringe
if (syr_grip[current_index]==1||((syr_rack[current_index]==8)||(syr_rack[current
  if (syr_size[current_index]==10)
    ringe = 1
  e__e
    syringe = 2
  endif
  grip = 1
  nest = 1
  ;if syringe in restart dont pick up
  if (syr_rack[current_index]!=8&&syr_rack[current_index]!=9)
    call puthold.tcl
    call stopchek.tcl
  endif
  if (syr_size[current_index]==10)
    syringe = 1
  else
    syringe = 2
  endif
  grip = 2
  nest = 1
  call gethold.tcl
  syr_grip[current_index] = 2
  call stopchek.tcl
endif

;squeeze out solution in syringe
if ((syr_size[current_index]==10&&syr_amount[current_index]>drop_amount1)||(syr_
  c''l waste.tcl
```

```
  call stopchek.tcl
endif
restart_info$[infoindex] = "Wash:  Emptied syringe"
savevars 7,"restart.dat"
infoindex = infoindex + 1

;dispense buffer
restart_info$[infoindex] = "Wash:  Started dispensing buffer"
savevars 7,"restart.dat"
infoindex = infoindex + 1
buf_amount = prot_vol[syr_process[current_index]]
if (syringe==2)
  buf_amount = buf_amount/division_factor
endif
buf_number = buf_place
run disp_buf.tcl
sol_avail[buf_place] = sol_avail[buf_place] - buf_amount
savevars 1,"amino.dat"
if (log_robot)
  logrobot$ = "dispense : " + numtostr$(buf_amount) + " nozzle: " + numtostr$(bu
  call logrobot.tcl
  logrobot$ = "solution : " + prot_buf$[syr_process[current_index]]
  call logrobot.tcl
endif
restart_info$[infoindex] = "Wash:  Finished dispensing buffer"
savevars 7,"restart.dat"
infoindex = infoindex + 1

;aspirate buffer
restart_info$[infoindex] = "Wash:  Started aspirating buffer"
savevars 7,"restart.dat"
infoindex = infoindex + 1
cup = 1
if (syr_size[current_index]==10)
  syringe = 1
else
  syringe = 2
endif
asp_amount = buf_amount
air_amount = prot_air[syr_process[current_index]]
if (syringe==2)
  air_amount = asp_amount/division_factor
endif
call syr_asp.tcl
restart_info$[infoindex] = "Wash:  Finished aspirating buffer"
savevars 7,"restart.dat"
infoindex = infoindex + 1
call stopchek.tcl

;rotate
rot_time = 60*prot_time[syr_process[current_index]]
call rotate.tcl
call stopchek.tcl

;squeeze out solution in syringe
```

```
call waste._:l
restart_info$(infoindex) = "Wash:  Emptied syringe"
savevars 7,"restart.dat"
infoindex = infoindex + 1
ca  .stopchek.tcl
```

```
;
;       Procedure: WASHING
;       Purpose  : executes robot actions for a washing
;       Author   : MJB
;

local a,buf_place,infoindex

infoindex = 1

;find buffer in reagent table
sarrayfind buf_place,contents$[1],prot_buf$[syr_process[current_index]]

;if syringe just started, get from input rack
if (syr_rack[current_index]!=8&&syr_rack[current_index]!=10&&syr_rack[current_in
  if (syr_size[current_index]==10)
    rack_type = 1
  else
    rack_type = 2
  endif
  rack_location = syr_rackloc[current_index]
  call getrack.tcl
  syr_grip[current_index] = 1
  call stopchek.tcl
endif

;switch grip on syringe
if (syr_grip[current_index]==1||((syr_rack[current_index]==8)||(syr_rack[current
  if (syr_size[current_index]==10)
     yringe = 1
  e__e
    syringe = 2
  endif
  grip = 1
  nest = 1
  ;if syringe in restart dont pick up
  if (syr_rack[current_index]!=8&&syr_rack[current_index]!=9)
    call puthold.tcl
    call stopchek.tcl
  endif
  if (syr_size[current_index]==10)
    syringe = 1
  else
    syringe = 2
  endif
  grip = 2
  nest = 1
  call gethold.tcl
  syr_grip[current_index] = 2
  call stopchek.tcl
endif

;squeeze out solution in syringe
if ((syr_size[current_index]==10&&syr_amount[current_index]>drop_amount1)||(syr_
  c-'l waste.tcl
```

```
  call stopchek.tcl
endif
restart_info$[infoindex] = "Wash:  Emptied syringe"
savevars 7,"restart.dat"
infoindex = infoindex + 1

;dispense buffer
restart_info$[infoindex] = "Wash:  Started dispensing buffer"
savevars 7,"restart.dat"
infoindex = infoindex + 1
buf_amount = prot_vol[syr_process[current_index]]
if (syringe==2)
  buf_amount = buf_amount/division_factor
endif
buf_number = buf_place
run disp_buf.tcl
sol_avail[buf_place] = sol_avail[buf_place] - buf_amount
savevars 1,"amino.dat"
if (log_robot)
  logrobot$ = "dispense : " + numtostr$(buf_amount) + " nozzle: " + numtostr$(bu
  call logrobot.tcl
  logrobot$ = "solution : " + prot_buf$[syr_process[current_index]]
  call logrobot.tcl
endif
restart_info$[infoindex] = "Wash:  Finished dispensing buffer"
savevars 7,"restart.dat"
infoindex = infoindex + 1

;aspirate buffer
restart_info$[infoindex] = "Wash:  Started aspirating buffer"
savevars 7,"restart.dat"
infoindex = infoindex + 1
cup = 1
if (syr_size[current_index]==10)
  syringe = 1
else
  syringe = 2
endif
asp_amount = buf_amount
air_amount = prot_air[syr_process[current_index]]
if (syringe==2)
  air_amount = asp_amount/division_factor
endif
call syr_asp.tcl
restart_info$[infoindex] = "Wash:  Finished aspirating buffer"
savevars 7,"restart.dat"
infoindex = infoindex + 1
call stopchek.tcl

;rotate
rot_time = 60*prot_time[syr_process[current_index]]
call rotate.tcl
call stopchek.tcl

;squeeze out solution in syringe
```

133 -

```
call waste.tcl
restart_info$[infoindex] = "Wash:  Emptied syringe"
savevars 7,"restart.dat"
inf~index = infoindex + 1
ca.  /stopchek.tcl
```

```
;------------------------------------------------------------------
;       Procedure: WASTE
;       Purpose  : move to waste and dispense liquid
;       Author   : MJB
;------------------------------------------------------------------

if (log_robot)
  logrobot$ = "waste : "+ numtostr$(syr_amount(current_index))
  call logrobot.tcl
endif

if (simulate_robot)
  stop
endif

request "ROBOT"

ROBOT.speed sp_med

ROBOT.move "waste"

ROBOT.speed sp_mslow

ROBOT.jog_s 0,0,-waste_jog
ROBOT.finish

/* Program to squeeze out liquid */
if (syringe==1)
  motor_steps = (syr_amount(current_index)+1)*syrg1_cal_dsp
else
  motor_steps = (syr_amount(current_index)+.25)*syrg2_cal_dsp
endif
if ((motor_steps-int(motor_steps))>=.5)
  motor_steps = int(motor_steps) + 1
else
  motor_steps = int(motor_steps)
endif
direction = 1
call strkmtr.tcl
syr_amount(current_index) = 0
savevars 5,"syringes.dat"
/*-------------------------------------*/

/* Program to suck up drop */
if (syringe==1)
  motor_steps = drop_amount1*syrg1_cal_asp
  syr_amount(current_index) = drop_amount1
else
  motor_steps = drop_amount2*syrg2_cal_asp
  syr_amount(current_index) = drop_amount2
endif
if ((motor_steps-int(motor_steps))>=.5)
  motor_steps = int(motor_steps) + 1
else
  motor_steps = int(motor_steps)
```

```
endif
direction = 0
call strkmtr.tcl
/*------------------------------------*/

ROBOT.speed sp_mslow

ROBOT.jog_s 0,0,waste_jog

free
```

```
;-----------------------------------------------------------
;       Procedure: WHATTODO
;       Purpose  : finds current_index
;       Author   : PJP
;-----------------------------------------------------------

local i, j, k, temp, temp_time, temp_busy
local new_busy, new_step, new_amino_count, new_amino_step
local first_temp, bail_out, new_syr_start[100], new_syr_end[100]

; set the current time (leave it dummied if we are simulating times)
if (!dont_wait)
  time_now = elapsed(0)
endif

current_index = 0
first_temp = 0
bail_out = 0

if (restart_index >= next_syringe)
  tell "Make sure that the temporary nests are empty!"
  restart_index = 0
endif

; if there are no syringes in process, start the first one or bail out
if (next_syringe <= 1)
  if (prot_proc[syr_process[next_syringe]] == step_ready && next_syringe <= num_
    current_index = next_syringe
    show "point 1"
  endif
  stop
endif

; If one was interrupted, continue that syringe
if (restart_index)
  current_index = restart_index
  show "point 2"
  restart_index = 0
  stop
endif

; Search through the active syringes and find the one with the shortest wait
first_temp = 1
i = 2
while (i < next_syringe)
  if (syr_processtime[i] < syr_processtime[first_temp])
    first_temp = i
  endif
  i = i + 1
endwhile

; Is the syringe with the shortest wait time ready to be processed?
if (syr_processtime[first_temp] <= time_now)
  current_index = first_temp
  stop
```

```
endif

; Are there no syringes ready in the input tray?  If there are none, no
; syringe will be processed at this time
if (prot_proc[syr_process[next_syringe]] != step_ready || next_syringe > num
  if (!dont_wait)
    current_index = 0
    stop
  else
    ; Simulate the next syringe
    current_index = first_temp
    show "point 3"
    time_now = syr_processtime[first_temp]
    stop
  endif
endif

;-------------------------------------------------------------
; Figure out if we can start the next syringe in the input rack
;-------------------------------------------------------------

;first, set up a temp array of current tumble end-times
i = 1
while (i < next_syringe)
  temp_processtime[i] = syr_processtime[i]
  temp_amino_step[i] = syr_amino_step[i]
  temp_step[i] = syr_process[i]
  i = i + 1
endwhile

; now set up arrays of 'robot busy' time slots using all active syringes
; set the index of the START and END time arrays for robot busy periods
temp_busy = 0
robot_busy_start[0] = time_now
robot_busy_end[0] = time_now
first_temp = 1

; while we haven't examined every active syringe's future, fill in the
; busy time slots
while (first_temp)

  temp_busy = temp_busy + 1

  ; Search through the active syringes and find the one with the shortest wait
  ; that hasn't been processed to its finish step yet
  first_temp = 0
  i = 1
  while (i < next_syringe)
    if (prot_proc[temp_step[i]] != step_finished)
      if (!first_temp)
        first_temp = i
      elseif (temp_processtime[i] < temp_processtime[first_temp])
        first_temp = i
      endif
    endif
```

```
      i = i + 1
    endwhile

    ; If we found a syringe that hasn't been processed to the end, process its
    ;   next steps
    if (first_temp)
      ; add up the adjacent robot move-times for that syringe
      temp = first_temp
      ; the START of these robot moves
      robot_busy_start[temp_busy] = temp_processtime[temp]
      temp_step[temp] = temp_step[temp] + 1
      temp_time = temp_processtime[temp]
      while (!(prot_choice[prot_proc[temp_step[temp]]] & param_stop))
        ;first, check if we are starting another amino
        if (prot_proc[temp_step[temp]] == step_nextamino)
          if (temp_amino_step[temp] < syr_amino_count[temp])
            temp_amino_step[temp] = temp_amino_step[temp] + 1
            temp_step[temp] = 1
          else
            temp_step[temp] = temp_step[temp] + 1
          endif
        elseif (prot_proc[temp_step[temp]] == step_stopifnofinaldep)
          if (syr_dodeprot$[temp] == "Y")
            temp_step[temp] = temp_step[temp] + 1
          else
            temp_step[temp] = num_procs
          endif
        else
          temp_step[temp] = temp_step[temp] + 1
          temp_time = temp_time + prot_robottime[temp_step[temp]]
        endif
      endwhile
      robot_busy_end[temp_busy] = temp_time          ;the END of thes robot moves

      ; re-assign its NEXT scheduled tumble end-time
      if (prot_proc[temp_step[temp]] != step_finished)
        temp_processtime[temp] = temp_time + (prot_time[temp_step[temp]] * 60)
      endif

    endif
  endwhile

  ; Now calculate the time slots needed by the next syringe in the input rack
  new_step = 1
  temp_time = time_now
  new_busy = 0
  new_amino_step = 1

  while (prot_proc[new_step] != step_finished)

    new_busy = new_busy + 1
    ; the START of these robot moves
    new_syr_start[new_busy] = temp_time
    while (!(prot_choice[prot_proc[new_step]] & param_stop))
      ;first, check if we are starting another amino
```

```
      if (pro._proc(new_step) == step_nextamino)
        if (new_amino_step < syr_amino_count(next_syringe))
          new_amino_step = new_amino_step + 1
          new_step = 1
        else
          new_step = new_step + 1
        endif
      elseif (prot_proc(new_step) == step_stopifnofinaldep)
        if (syr_dodeprot$(next_syringe) == "Y")
          new_step = new_step + 1
        else
          new_step = num_procs
        endif
      else
        new_step = new_step + 1
        temp_time = temp_time + prot_robottime(new_step)
      endif
    endwhile
    ; the END of these robot moves
    new_syr_end(new_busy) = temp_time

    ; re-assign its NEXT scheduled tumble end-time
    if (prot_proc(new_step) != step_finished)
      temp_time = temp_time + (prot_time(new_step) * 60)
      new_step = new_step + 1
    endif

endwhile

; Now compare the time line of the active syringes and the new syringe
; and see if the new syringe fits
i = 1
j = 0
if (new_syr_end(1) < robot_busy_start(1)) ;the first block of moves is clear
  i = 2
  while ((i <= new_busy) && (!bail_out) && (j <= temp_busy))

    if ((new_syr_start(i) >= robot_busy_end(j)) && (new_syr_end(i) <= robot_busy
      i = i + 1
    elseif ((new_syr_start(i) >= robot_busy_start(j)) && (new_syr_start(i) < rob
      bail_out = i
      k = j
    elseif ((new_syr_end(i) > robot_busy_start(j)) && (new_syr_end(i) <= robot_b
      bail_out = i
      k = j
    elseif ((robot_busy_start(j) >= new_syr_start(i)) && (robot_busy_start(j) <
      bail_out = i
      k = j
    elseif ((robot_busy_end(j) > new_syr_start(i)) && (robot_busy_end(j) <= new_
      bail_out = i
      k = j
    else
      j = j + 1
    endif
```

```
  endwhile
else
  bail_out = 5
endif

;if 'next_syringe' not on hold, and there is one in the input rack, get it
if (!bail_out)
  current_index = next_syringe
  show "point 4"
else
  ; if don't want to wait and there is a syringe in process
  current_index = 0
  if (dont_wait)
    time_now = time_now + 60
  endif
endif
```

```
;=====================================================
;        Procedure: FINDTOP
;        Purpose  : turn on motor until sensor is triggered
;        Author   : MJB
;=====================================================

if (simulate_robot)
  stop
endif

;load zero into plc counter register
SHARK.writebit o_clear_counter,1
delay 100
SHARK.writebit o_clear_counter,0
;==================================

;set counter enable
SHARK.writebit o_start_counter,1
;==================================

;set direction to forward and turn on motor (motor will turn off
automatically)
SHARK.writebit 224,1
delay 500
SHARK.writebit 401,1
;==================================

;wait for syringe to be sensed
SHARK.readbit i_sense_syringe,a
timer findtop_time
while (!a)
  SHARK.readbit i_sense_syringe,a
  if (elapsed(findtop_time)>13)

    ;stop motor
    SHARK.writebit 401,0
    delay 500

    ;home motor
    call homemtr.tcl
    while (!home_done)
    endwhile
    delay 250

    ;load zero into plc counter register
    SHARK.writebit o_clear_counter,1
    delay 100
    SHARK.writebit o_clear_counter,0
    ;==================================

    ;call error
    error_code = 5
    call error.tcl
```

```
    ;restart timer
    timer findtop_time

    ;set counter enable
    SHARK.writebit o_start_counter,1
    ;================================

    ;start motor to find top of syringe
    SHARK.writebit 224,1
    delay 500
    SHARK.writebit 401,1

  endif
endwhile
;==============================

;read the number of counts to reach the syringe
SHARK.readctr i_motor_pulses,motor_pulses
show "motor pulses = " + numtostr$(motor_pulses)
;============================================

;disable counter
SHARK.writebit o_start_counter,0
;=============================
```

CLAIMS:

1. A system for the solid phase synthesis of multiple species of organic compounds each formed by repeated synthetic cycles of synthetic steps, said system comprising:

a computer for processing a program of instructions correlated to predetermined synthetic cycles for the synthesis of each of said multiple species;

an automated robot responsive to said computer, said automated robot operative to cause said synthesis steps to be performed in accordance with the synthetic cycles of each of said multiple species; and

a timing protocol, implemented by said program of instruction and executed by said computer, for directing synthetic steps of at least two different species to be performed concurrently.

2. The system of claim 1 wherein each of said species has an associated sequence specific timing protocol, the compilation of all sequence specific timing protocols of species currently under synthesis being a cumulative remaining timing protocol, said system further including means for determining whether there is a timing conflict between the sequence specific timing protocol of a new desired species and the cumulative remaining timing protocol.

3. The system of claim 1 wherein said synthetic steps include the steps of washing, adding deprotection reagents, deprotection, adding coupling reagents and coupling.

4. The system of claim 1 wherein said synthetic steps include active and passive synthetic steps, said

timing protocol directing said active steps within a first synthesis of a first species to be performed concurrently with passive steps within a second synthesis of a second species.

5. The system of claim 1 wherein said organic compounds includes peptides.

6. The system of claim 1 further comprising:
a first set of a plurality of syringes, each syringe containing resin for repeatably coupling compounds to a solid support so as to synthesize a first species; and
a second set of a plurality of syringes, each syringe containing a desired compound that is to be coupled to the solid support during one of said synthetic cycles.

7. The system of claim 6 where said compound that is to be coupled is an amino acid for the synthesis of a peptide.

8. The system of claim 7 wherein said solid phase synthesis is a Merrifield solid phase peptide synthesis.

9. The system of claim 1 wherein said automated robot includes
a gripper arm responsive to said computer, said gripper arm selectively positioning and manipulating syringes from said first and second sets of a plurality of syringes to thereby aspirate or dispense solvents and regents required within said synthetic cycles.

10. The system of claim 1 further comprising a database containing parameter variables required for the synthesis of said multiple species of organic compounds, said database including the characteristic sequence of the repeated synthetic cycles for each of said multiple species.

11. A system for the solid phase synthesis of multiple peptides each formed by repeated synthetic cycles of synthetic steps, said system comprising:

a first set of syringes each holding a solid support;

a second set of syringes each holding a desired amino acid, said first set of syringes serving as the reaction vessels for coupling desired amino acids from said second set of syringes to the solid supports;

a computer for processing a program of instructions correlated to predetermined synthetic cycles for the synthesis of multiple peptides; and

an automated robot responsive to said computer, said automated robot operatively coupled to said first and second sets of syringes for aspirating and dispensing reagents so as to cause said repeated synthesis cycles to be performed in accordance with a desired peptide synthesis to thereby repeatably couple a desired amino acid to a solid support, said repeated synthesis cycles including the synthetic steps of washing, adding deprotection reagents, deprotection, adding coupling reagents and coupling,

said repeated synthesis cycles performed in accordance with a timing protocol wherein said washing, adding deprotection reagents or adding coupling reagents is performed concurrently with said deprotection or coupling for different peptide syntheses.

12. The system of claim 11 wherein said first set of syringes is made of polypropylene.

13. The system of claim 11 wherein each of said first set of syringes includes a solid phase retaining device

14. The system of claim 13 wherein said solid phase retaining device is a frit.

15. The system of claim 11 wherein said solid support comprises a resin selected from the group of polystyrene and polyethylene glycol.

16. The system of claim 11 where said automated robot includes an arm operatively connected to said robot so as to manipulate each of said first and second sets of syringes.

17. The system of claim 16 wherein said first and second sets of syringes each includes a plunger operatively coupled thereto, said plunger movable by said arm of said automated robot to thereby aspirate or dispense reagents.

18. The system of claim 17 wherein the content amount of reagents in each of said syringes is determined by the plunger position of the corresponding syringe.

19. The system of claim 11 further including an optical sensor responsive to the movement of said first and second sets of syringes, said optical sensor confirming the positioning of said first and second sets of syringes.

20. The system of claim 11 wherein said synthesis of peptides employs Fmoc/tBu chemistry.

21. The system of claim 11 further comprising piston pumps and delivery cups, said piston pumps delivering solvents and reagents to said delivery cups from which said first and second sets of syringes dispense or aspirate said solvents or reagents.

22. The system of claim 11 further including a database containing parameter variables required for the synthesis of the multiple peptides.

23. A process for the solid phase automated synthesis of multiple species of organic compounds formed by repeated synthetic cycles of synthetic steps, said process comprising the steps of:

a) initiating a synthetic step in a synthetic cycle of a species;

b) determining if the time for said synthetic step of step a) is sufficient for initiating a synthetic step in the synthetic cycle of a different species, and if so,

c) initiating a synthetic step in the synthetic cycle of said different species; and

d) repeating steps a), b) and c) until said multiple species are synthesized.

24. The process of claim 23 wherein said synthetic steps include washing, deprotection and coupling for the synthesis of multiple peptides.

25. The process of claim 24 wherein said deprotection and coupling are performed in a plurality of syringes which serve as reaction vessels.

26. A process for the automated solid phase synthesis of multiple peptides formed by repeated cycles of washing, adding deprotection reagents, deprotection, adding coupling reagents and coupling so as to repeatably couple amino acids to a solid support, said process comprising the steps of:

a) filling a first set of syringes, each with a resin solid support;

b) filling a second set of syringes, each with a desired amino acid;

d) washing one of said first set of syringes with a first solution;

e) adding deprotection solution to the syringe of step d) to initiate deprotection;

f) while deprotection of step e) is proceeding repeating steps d) and e) if the time for performing those steps is less than the time required for deprotection, otherwise proceeding with step g);

g) adding coupling solution to the syringe of step d) to initiate coupling to thereby couple a desired amino acid to the solid support; and

h) while coupling of step g) is proceeding repeating steps f) and g) if the time for performing those steps is less than the time for coupling, otherwise proceeding with steps d) and e) until the time for coupling expires.

27. The process of claim 26 wherein steps a) through h) are repeated until a peptide is synthesized.

28. The process of claim 26 further comprising the step of washing each of said first set of syringes of step d) prior to adding coupling solution.

29. The process of claim 26 wherein the system is automated through the use of a robot, said robot adding coupling and deprotection solution by manipulating the syringes so as to aspirate and dispense reagents.

30. The process of claim 26 wherein the first solution is dimethylformamide.

31. The process of claim 26 wherein the deprotection solution is piperidine/DMF.

32. The process of claim 26 wherein the coupling solution is a solution of DIC/HOBt.

33. The process of claim 26 further comprising the step of interactively prompting a user for parameter variables thereby forming a database required for the synthesis of the multiple peptides, said database including the characteristic amino acid sequence of each of the multiple peptides.

34. The process of claim 26 further comprising the step of adding or deleting peptide sequences to a list of peptides to be synthesized.

35. A process for the solid phase automated synthesis of multiple species of organic compounds formed by repeated synthetic cycles of synthetic steps, said process comprising the steps of:

a) determining a sequence specific timing protocol associated with a desired species to be synthesized, said sequence specific timing protocol including when each synthetic step for the desired species occurs temporally;

b) determining the cumulative remaining timing protocol of species currently under synthesis, said cumulative remaining timing protocol including when each synthetic step for the species currently under synthesis occurs temporally; and

c) determining whether a timing conflict exists between said sequence specific timing protocol and said cumulative remaining timing protocol, a timing conflict occurring when one synthetic step for the desired species to be synthesized cannot be performed concurrently with the synthetic steps of species currently under synthesis.

36. The process of claim 35 further comprising the step of time shifting the synthesis initiation of the desired species to be synthesized so as to resolve the time conflict.

37. The process of claim 35 wherein a time conflict occurs when an active synthetic step of the desired species to be synthesized is temporally coincident with any active synthetic steps for the species currently under synthesis.

38. The process of claim 35 wherein said organic compounds includes peptides.

39. The process of claim 35 further comprising the step of entering parameter variables required for the synthesis of the organic compounds.

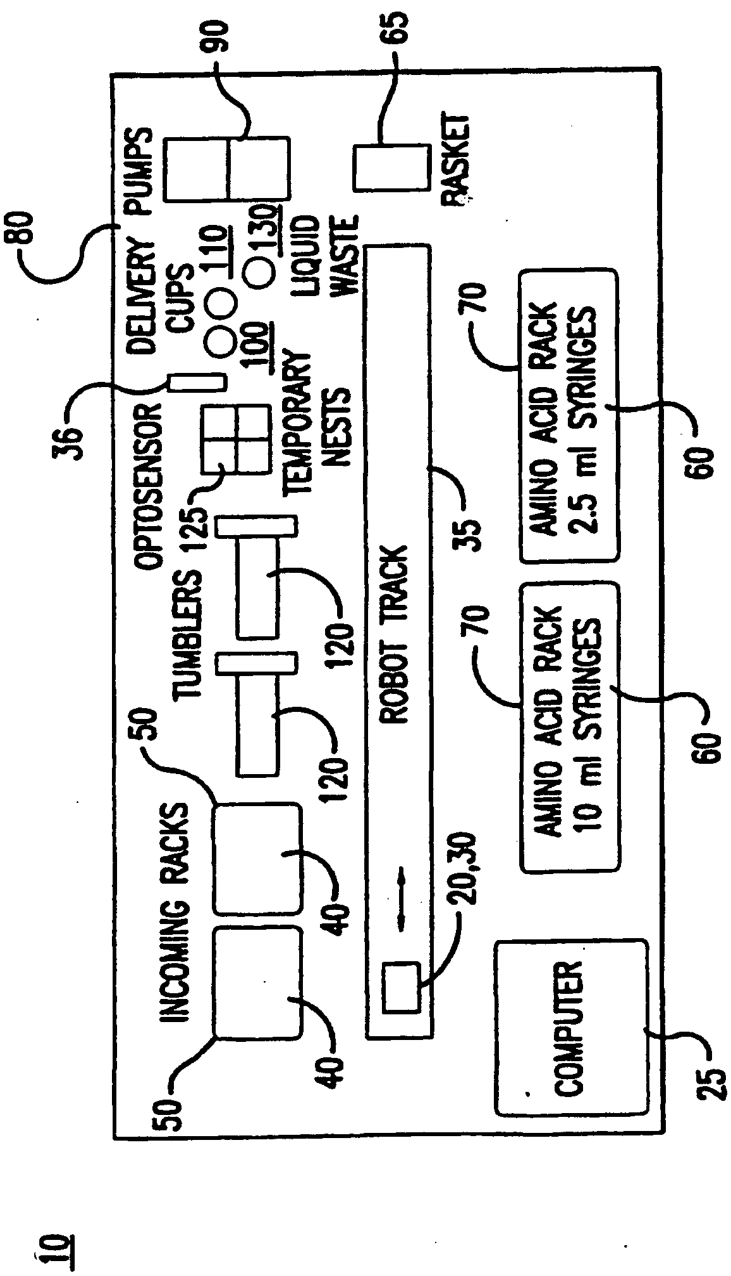

80
DELIVERY PUMPS
90
65
BASKET
CUPS
100
110
130
LIQUID WASTE
36
OPTOSENSOR
TEMPORARY NESTS
125
TUMBLERS
120
120
ROBOT TRACK
35
70
AMINO ACID RACK 2.5 ml SYRINGES
60
70
AMINO ACID RACK 10 ml SYRINGES
60
INCOMING RACKS
50
50
40
40
20,30
COMPUTER
25
10

FIG. 1

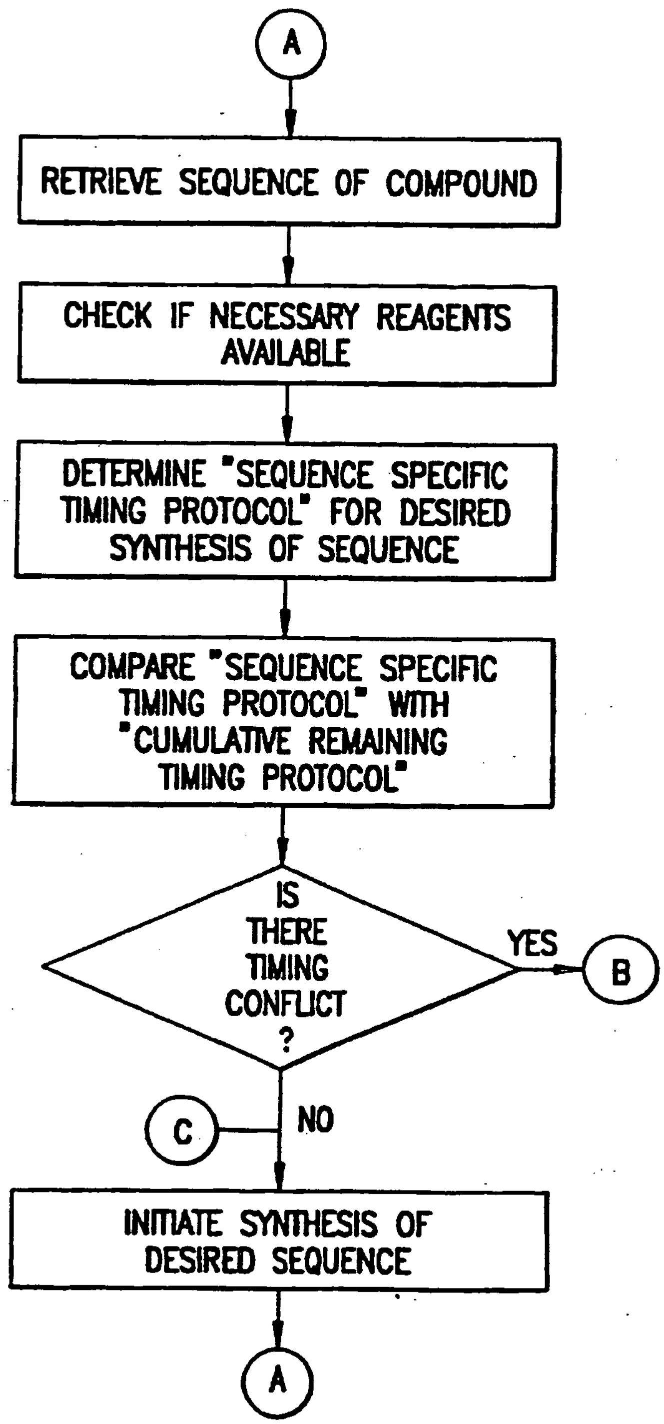

A
RETRIEVE SEQUENCE OF COMPOUND
CHECK IF NECESSARY REAGENTS AVAILABLE
DETERMINE "SEQUENCE SPECIFIC TIMING PROTOCOL" FOR DESIRED SYNTHESIS OF SEQUENCE
COMPARE "SEQUENCE SPECIFIC TIMING PROTOCOL" WITH "CUMULATIVE REMAINING TIMING PROTOCOL"
IS THERE TIMING CONFLICT ?
YES
B
NO
C
INITIATE SYNTHESIS OF DESIRED SEQUENCE
A

FIG. 2A

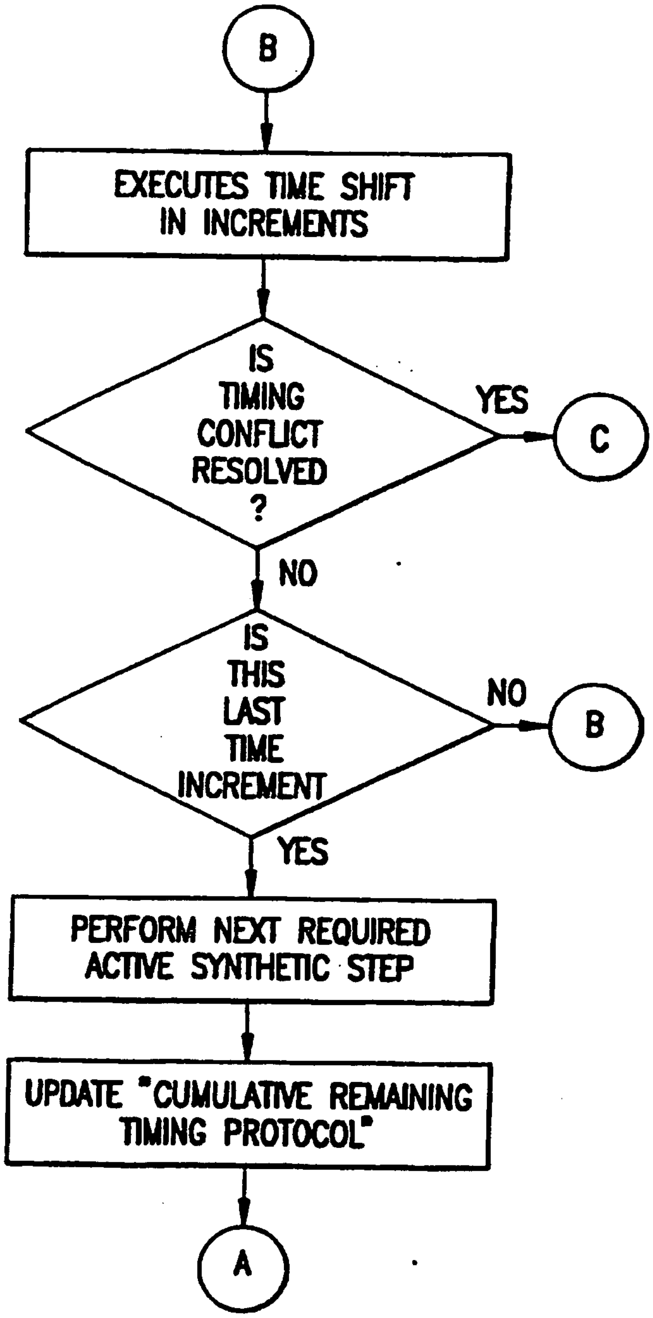

B
EXECUTES TIME SHIFT IN INCREMENTS
IS TIMING CONFLICT RESOLVED ?
YES
C
NO
IS THIS LAST TIME INCREMENT
NO
B
YES
PERFORM NEXT REQUIRED ACTIVE SYNTHETIC STEP
UPDATE "CUMULATIVE REMAINING TIMING PROTOCOL"
A

FIG. 2B

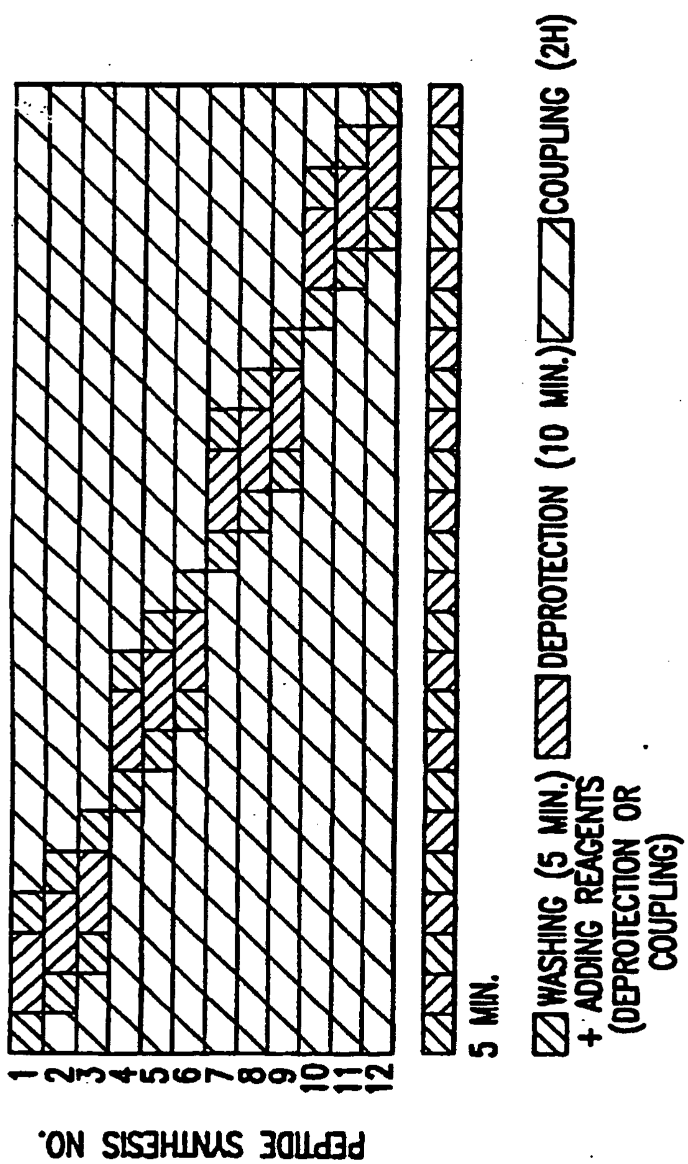

PEPTIDE SYNTHESIS NO.
1
2
3
4
5
6
7
8
9
10
11
12
5 MIN.
WASHING (5 MIN.) + ADDING REAGENTS (DEPROTECTION OR COUPLING)
DEPROTECTION (10 MIN.)
COUPLING (2H)

FIG.3

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US96/01168

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :Please See Extra Sheet.
US CL :422/116; 525/54.11; 530/334; 935/88
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 395/904, 906; 422/116; 525/54.1, 54.11; 528/312, 328; 530/334, 810; 935/88

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

APS, DIALOG
search terms: peptide, Merrifield, robot, computer, automated, programmable, simultaneous, concurrent

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US, A, 5,147,608 (HUDSON ET AL) 15 September 1992.	1-39
X	US, A, 5,368,823 (MCGRAW ET AL) 29 November 1994, see column 2, lines 3-8 and 47-54, column 4, lines 38-42 and 66-68, column 7, line 38 - column 8, line 34, column 9, line 51 - column 10, line 2, and Table 4.	1, 3-5, 10
A	EP, A, 0 529 504 (SHIMADZU CORPORATION) 03 March 1993.	1-39
A	Biochemical Society Transactions, Volume 20, issued 1992, Fox, "Automatic multiple peptide synthesis", pages 851-853.	1-39

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search

09 APRIL 1996

Date of mailing of the international search report

13 MAY 1996

Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231
Facsimile No. (703) 305-3230

Authorized officer
JEFFREY E. RUSSEL
Telephone No. (703) 308-0196

Form PCT/ISA/210 (second sheet)(July 1992)*

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US96/01168

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	Tetrahedron, Volume 45, Number 24, issued 1989, Schnorrenberg et al, "Fully Automatic Simultaneous Multiple Peptide Synthesis In Micromolar Scale - Rapid Synthesis Of Series Of Peptides For Screening In Biological Assays", pages 7759-7764, see page 7759, sixth paragraph, page 7761, first and second paragraphs, Tables 1 and 2, and Figure 1.	1, 3-5, 10
A	International Journal Of Peptide And Protein Research, Volume 40, issued 1992, Zuckermann et al, "Design, construction and application of a fully automated equimolar peptide mixture synthesizer", pages 497-506.	1-39

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US96/01168

A. CLASSIFICATION OF SUBJECT MATTER:
IPC (6):

B01J 8/02; C07K 1/04, 1/06, 1/08, 1/10; C08F 283/00; C08G 69/10; G05B 13/00, 19/402